DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

GENERATION OF NON-HOMOGENEOUS POISSON PROCESSES BY THINNING: PROGRAMMING CONSIDERATIONS AND COMPARISON WITH COMPETING ALGORITHMS

by

John Scott Redd

December 1978

Thesis Advisor: P.A.W. Lewis

Approved for public release; distribution unlimited.

T187397

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Generation of Non-Homogeneous Poisson Processes by Thinning: Programming Considerations and Comparison with Competing Algorithms

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis;
December 1978

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
John Scott Redd

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
December 1978

13. NUMBER OF PAGES
100

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Poisson processes
thinning algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this thesis we study several computer implementations of the thinning algorithm, a new method for generating non-homogeneous Poisson processes. The method, developed by Professor P.A.W. Lewis, Naval Postgraduate School, Monterey, California, and G.S. Shedler, IBM Research Laboratory, San Jose, California, is valid for Poisson processes with any given intensity function. The basic thinning algorithm is modified

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

(20. ABSTRACT Continued)

to exploit several refinements which reduce computer execution time by approximately one-third. The basic and modified thinning programs are compared with a previous algorithm of Lewis and Shedler, the Poisson decomposition and gap-statistics algorithm, which is easily implemented for Poisson processes with intensity functions of the form $\exp(a_0 + a_1 t + a_2 t^2)$. The thinning programs are competitive in both execution time and computer memory requirements. One program implementation generates the events in a Poisson process one at a time; another program implements the algorithmic refinements which improve efficiency.

Approved for public release; distribution unlimited.

Generation of Non-Homogeneous Poisson Processes by Thinning: Programming Considerations and Comparison with Competing Algorithms

by

John Scott Redd
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 1978

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93940

ABSTRACT

In this thesis we study several computer implementations of the thinning algorithm, a new method for generating non-homogeneous Poisson processes. The method, developed by Professor P.A.W. Lewis, Naval Postgraduate School, Monterey, California, and G.S. Shedler, IBM Research Laboratory, San Jose, California, is valid for Poisson processes with any given intensity function. The basic thinning algorithm is modified to exploit several refinements which reduce computer execution time by approximately one-third. The basic and modified thinning programs are compared with a previous algorithm of Lewis and Shedler, the Poisson decomposition and gap-statistics algorithm, which is easily implemented for Poisson processes with intensity functions of the form $\exp(a_0+a_1t+a_2t^2)$. The thinning programs are competitive in both execution time and computer memory requirements. One program implementation generates the events in a Poisson process one at a time; another program implements the algorithmic refinements which improve efficiency.

TABLE OF CONTENTS

I. INTRODUCTION ---------------------------------- 10

II. THE THINNING THEOREM ------------------------------- 15

III. ALGORITHMS CONSIDERED ------------------------------ 20

A. POISSON DECOMPOSITION AND GAP-STATISTICS ALGORITHM -- 20

1. Usage ------------------------------------ 20

2. Algorithm Statement ---------------------- 21

B. THE BASIC THINNING ALGORITHM ----------------- 22

1. Usage ------------------------------------ 22

2. Algorithm Statement ---------------------- 23

C. THE ONE-AT-A-TIME THINNING ALGORITHM --------- 23

1. Usage ------------------------------------ 23

2. Algorithm Statement ---------------------- 24

IV. METHODOLOGY FOR ALGORITHM COMPARISON ------------- 26

A. MEASURES OF EFFECTIVENESS ------------------- 26

1. Computational Speed ---------------------- 26

2. Computer Memory Requirements ------------- 28

B. MEASUREMENT CONSIDERATIONS ------------------ 28

C. TEST SETUP ---------------------------------- 29

1. Computational Speed ---------------------- 29

2. Computer Memory Requirements ------------- 30

3. Test Cases Utilized ---------------------- 31

V. EFFICIENCY AND PROGRAMMING CONSIDERATIONS -------- 38

A. GENERAL ------------------------------------- 38

B. UTILIZATION OF ARRAYS OF RANDOM VARIATES ----- 39

C. UTILIZATION OF INTENSITY FUNCTION LOWER BOUND ---------- 42

D. UTILIZATION OF EXPONENTIAL VARIATES FOR THINNING OF EXPONENTIAL POLYNOMIAL INTENSITY FUNCTIONS ---------- 43

E. RECYCLING OF THINNING VARIATES ---------- 45

1. Recycling of Uniform Variates ---------- 45

2. Recycling of Exponential Variates ---------- 47

F. FINAL PROGRAM ---------- 50

VI. RESULTS, CONCLUSIONS AND RECOMMENDATIONS ---------- 52

A. GENERAL ---------- 52

B. MEASURE OF EFFECTIVENESS RESULTS ---------- 54

1. The Basic Thinning Algorithm ---------- 54

2. The Modified Thinning Algorithm (Final Program) ---------- 54

3. The One-At-A-Time Thinning Algorithm ---------- 59

C. CONCLUSIONS ---------- 62

D. RECOMMENDATIONS ---------- 65

APPENDIX A: GENERATION OF DEGREE-ONE EXPONENTIAL POLYNOMIAL NON-HOMOGENEOUS POISSON PROCESSES ---------- 67

APPENDIX B: RESULTS OF EFFICIENCY MODIFICATIONS ---------- 69

COMPUTER PROGRAMS ---------- 76

LIST OF REFERENCES ---------- 98

INITIAL DISTRIBUTION LIST ---------- 99

LIST OF TABLES

I. Generation Times for Arrays of Shuffled Random Variates from LLRANDOM ------------------ 41

II. Comparison of Two Algorithms ------------------ 55

III. Comparison of Poisson Decomposition and Gap-Statistics and Basic Thinning Algorithms --- 56

IV. Comparison of Best Case for Both Algorithms ---- 58

V. Comparison of Poisson Decomposition and Gap-Statistics and One-At-A-Time Algorithms -------- 61

VI. Computer Memory Requirements ------------------ 63

VII. Comparison of Gap-Statistics and Thinning Algorithms for Generation of Processes with Degree-One Exponential Polynomial Intensity Functions ------------------------------------ 68

VIII. Lower Bound Versus No Lower Bound; Uniform Thinning Variates ------------------------------ 70

IX. Exponential Thinning Variates Versus Uniform Thinning Variates for Exponential Polynomial Intensity Functions ---------------------------- 71

X. Recycling Versus No Recycling; Uniform Thinning Variates ------------------------------ 72

XI. Recycling Versus No Recycling; Exponential Thinning Variates ------------------------------ 73

XII. Final Thinning Program Versus Basic Thinning Program ------------------------------ 75

LIST OF FIGURES

1. Sample Intensity Functions of Bounding and Desired Poisson Processes for Degree-Two Exponential Polynomial Case ------------------------- 19

2. Case I - Sample Intensity Function ------------------ 32

3. Case II - Sample Intensity Function ----------------- 33

4. Case III - Sample Intensity Function ---------------- 34

5. Case IV - Sample Intensity Function ----------------- 35

6. Case V - Sample Intensity Function ------------------ 36

7. Case VI - Sample Intensity Function ----------------- 37

ACKNOWLEDGMENT

I am deeply indebted to Professor Peter A.W. Lewis. His excellent classroom presentations of subjects in simulation were responsible for my initial interest in this area. As my thesis advisor, his guidance, assistance and interest were, of course, invaluable.

I am particularly pleased to give richly deserved recognition to Donna, my wonderful wife. Her support has always been fundamental to the success of my undertakings.

Finally, none of this would have been possible without the graciousness of Him who knows all things with probability one.

I. INTRODUCTION

The Poisson process is a widely known and studied stochastic process. It is frequently used to describe random arrivals at some type of service facility such as a service station fuel pump or a bank teller's window. In its most common form, the "rate" of these arrivals is considered to be constant over time. This is the homogeneous Poisson process which has the familiar property that times between arrivals (or events) are exponentially distributed with mean equal to the inverse of the rate.

The assumption of a constant rate, or homogeneity, is at best tenuous when applied to real world data. For example, the rate of arrivals at a traffic light typically varies from very high during rush hours to very low in the early morning. In addition to this cyclic time-of-day effect, arrival rates may exhibit longer term increases or decreases. Further, these effects may be superimposed upon shorter term effects to produce a more complex rate which varies with time. These processes for which arrival rates vary with time may often be represented by a non-homogeneous Poisson process, that is, a Poisson process with a time dependent rate of arrival.

The generic term of Poisson process includes then both homogeneous and non-homogeneous Poisson processes. LEWIS and SHEDLER [Ref. 1] define the Poisson process generally in

terms of a monotone non-decreasing right-continous function $\Lambda(t)$ which is bounded in any finite interval. Then the number of points, $N(t'',t')$, in any finite interval has a Poisson distribution with parameter $\mu(t'',t') = \Lambda(t') - \Lambda(t'')$. Thus, for example in $(0,t']$, with $t' \geq 0$, $P\{N(t'',t') = n\} \equiv P\{N_{t'}=n\} = \mu_0^n e^{-\mu_0}/n!$, where $\mu_0 = \mu(0,t') = \Lambda(t') - \Lambda(0)$.

The right derivative $\lambda(t)$ of $\Lambda(t)$ will be assumed to exist and is called the rate function or intensity function of the process. $\Lambda(t)$ is called the integrated rate function and has the interpretation that for $t \geq 0$, $\Lambda(t) - \Lambda(0) = E[N_t]$. For the homogeneous Poisson process $\lambda(t)$ is a constant, e.g. λ, and thus the integrated rate function is simply the product of λ and t, i.e. the expected value of $N_t \equiv N(0,t)$.

While simulation of homogeneous Poisson processes is relatively straightforward, the non-homogeneous Poisson process is more problematical. Times between events are not exponential in the general case and simulation has typically been tailored to specific classes of intensity functions. LEWIS and SHEDLER [Ref. 1] list three general methods for simulating non-homogeneous Poisson processes and one method for a special rate function. The general methods include the time scale transformation method and the conditioning and order-statistics method. The special method is the gap-statistics method, a method which is particular to the degree-one exponential polynomial intensity fucntion, i.e. those of the form $\lambda(t) = \exp(b_0 + b_1 t)$.

Implementation of the general methods on a computer may pose special problems. Often the inverse of the integrated rate function is not explicit and must be computed numerically. Other problems in implementation generally result in lower efficiency, as measured by execution time or computer storage requirements or both.

One class of intensity functions which is of general interest is the degree-two exponential polynomial family. That is, those with intensity function of the form $\lambda(t) = \exp(a_0 + a_1 t + a_2 t^2)$. This family of functions has the property of being positive for all values of t, a necessary condition for an intensity function. Additionally, by varying the magnitude and sign of the coefficients, the exponential polynomial of degree two can be made to be monotone increasing or decreasing over time, as well as increasing and then decreasing, or vice versa. Use of this type of intensity function also leads to statistical procedures which are relatively simple.

LEWIS and SHEDLER [Ref. 2] proposed a new method of generating the non-homogeneous Poisson process with degree-two exponential polynomial intensity function. It involves decomposition of the degree-two exponential polynomial intensity function, $\lambda(t)$, into two functions, a degree-one exponential polynomial function, $\lambda_L(t)$, and a difference function, $\lambda_D(t) = \lambda(t) - \lambda_L(t)$. This procedure allows the points in the degree-one exponential polynomial event stream

to be generated using the gap-statistics method, which is highly efficient when implemented on a computer. The remaining points with intensity function $\lambda_D(t)$ are generated by other methods and then merged with the other events.

PATROW [Ref. 3] implemented two algorithms, the time scale transformation algorithm and the Poisson decomposition and gap-statistics technique, and compared them for computational speed and computer memory requirements. His results indicated that the Poisson decomposition and gap-statistics technique was from two to seven times faster than the time scale transformation algorithm, although the former required about thirty percent more computer memory.

PATROW's work [Ref. 3] is also an excellent self-contained reference on Poisson processes, bringing many references together under one cover.

A recent result of LEWIS and SHEDLER [Ref. 1] develops a new method for generation of points in a non-homogeneous Poisson process. This method, called "thinning", is similar to the general conditioning-acceptance-rejection method but has subtle differences which are computationally significant. The thinning method is straightforward in both an analytical and a computational sense, and is valid for any type of intensity function. The thinning theorem is presented in Section II.

This thesis is, in a sense, a sequel to PATROW's work [Ref. 3]. Its purpose is to implement the thinning algorithm

in computer program form and to compare it to the Poisson decomposition and gap-statistics algorithm implemented by PATROW [Ref. 3]. The latter implementation was designed for a specific subset of intensity functions, degree-two exponential polynomials. Since the Poisson decomposition and gap-statistics method outperformed a general case algorithm (time scale transformation) by a significant margin, comparing the thinning method to the Poisson decomposition and gap-statistics method should give a reasonable indication of the thinning algorithm's performance in generating non-homogeneous Poisson processes with other than degree-two exponential polynomial intensity functions.

Section III lists the two algorithms considered, as well as a special application of the thinning process which will be of interest to those involved in event-step simulation. Section IV describes the methodology used in comparing the algorithms while Section V deals with aspects of the thinning procedure which may be exploited to enhance its overall effectiveness in a variety of situations. Finally, Section VI presents the results and conclusions of the comparisons of the algorithms. Appendices A and B contain secondary results and computer program listings following the appendices.

II. THE THINNING THEOREM

The underlying concept of the thinning method involves the use of a "bounding" Poisson process, $\{N_t^* : t \geq 0\}$, where N_t^* is the number of points in the bounding process in the interval $(0,t]$. This process may be either homogeneous or non-homogeneous Poisson, but should be one which is easy to simulate on a computer. It is called bounding because its intensity function, denoted $\lambda^*(t)$, bounds the intensity function $\lambda(t)$, of the nonhomogeneous Poisson process which is to be simulated over the fixed interval $(0,t']$. That is, $\lambda^*(t) \geq \lambda(t)$ for all t in $(0,t']$. Points at T_i^*, $i = 1, \ldots, N_{t'}^*$, are generated for the bounding process over the interval $(0,t']$. These points are then deleted, or "thinned", with independent probabilities equal to $1 - (\lambda(T_i^*)/\lambda^*(T_i^*))$. Thus the probability that a point of the bounding process, T_i^*, is a point of the process being generated is equal to the ratio of the intensity functions evaluated at that point, i.e. $\lambda(T_i^*)/\lambda^*(T_i^*)$.

More formally:

Theorem 1. Consider the one-dimensional non-homogeneous Poisson process $\{N_t^* : t \geq 0\}$ with rate function $\lambda^*(t)$. The number of events, $N_{t'}^*$, in the fixed interval $(0,t']$ has a Poisson distribution with parameter $\mu^*(0,t') = \mu^* = \Lambda^*(t') - \Lambda^*(0)$.

Let T_1^*, T_2^*, T_3^*, ..., $T_{N_{t'}^*}^*$, be the times of the events of the process in the interval $(0,t']$.

Suppose that for $0 \leq t \leq t'$, $\lambda(t) \leq \lambda^*(t)$. For $i = 1, 2, \ldots, N_t^*$, delete the event at T_i^* with independent probability $1 - \lambda(T_i^*)/\lambda^*(T_i^*)$.

Then the remaining times form a non-homogeneous Poisson process with rate function $\lambda(t)$ in the interval $(0,t']$.

Proof:

We assume that $\lambda(t)$ is continuous and use the definition of the Poisson process based on incremental probabilities. Thus we need to show that the occurrence of an event in $(t,t+dt]$ is independent of the number or times of occurrence of events before t, and that

$$P\{N_{t+dt} - N_t = 0\} = 1 - \lambda(t)dt + o(dt),$$

$$P\{N_{t+dt} - N_t = 1\} = \lambda(t)dt + o(dt),$$

and

$$P\{N_{t+dt} - N_t > 1\} = o(dt).$$

Now we have that

P{no event from $\{N_t : t \geq 0\}$ in (t,t+dt]}

= P{no event from $\{N_t^* : t \geq 0\}$ in (t,t+dt]}+ P{event from $\{N_t^* : t \geq 0\}$ in (t,t+dt] and it is "thinned"}

$$= 1 - \lambda^*(t)dt + [\lambda^*(t)dt]\cdot[1 - \lambda(t)/\lambda^*(t)] + o(dt)$$

$$= 1 - \lambda^*(t)dt + \lambda^*(t)dt - \lambda^*(t)\cdot\lambda(t)/\lambda^*(t)\cdot dt + o(dt)$$

$$= 1 - \lambda(t)dt + o(dt).$$

Similarly:

P{one event from $\{N_t : t \geq 0\}$ in (t,t+dt]}

= P{event from $\{N_t^* : t \geq 0\}$ in (t,t+dt] which is not "thinned"}

$$= \lambda^*(t)dt\cdot\lambda(t)/\lambda^*(t) + o(dt)$$

$$= \lambda(t)dt + o(t)$$

Also it follows directly that

$$P\{\text{more than one event in } (t,t+dt]\} = o(dt)$$

Moreover, an event in (t,t+dt] is independent of what happens before t because:

1. $\{N_t^* : t \geq 0\}$ is a Poisson process and therefore has independent increments, and

2. The thinning uniform random variate is independent of other thinning variates, and is independent of the Poisson process $\{N_t^* : t \geq 0\}$.

Q.E.D.

Figure 1 shows a graphical representation of a particular case of bounding and object intensity functions.

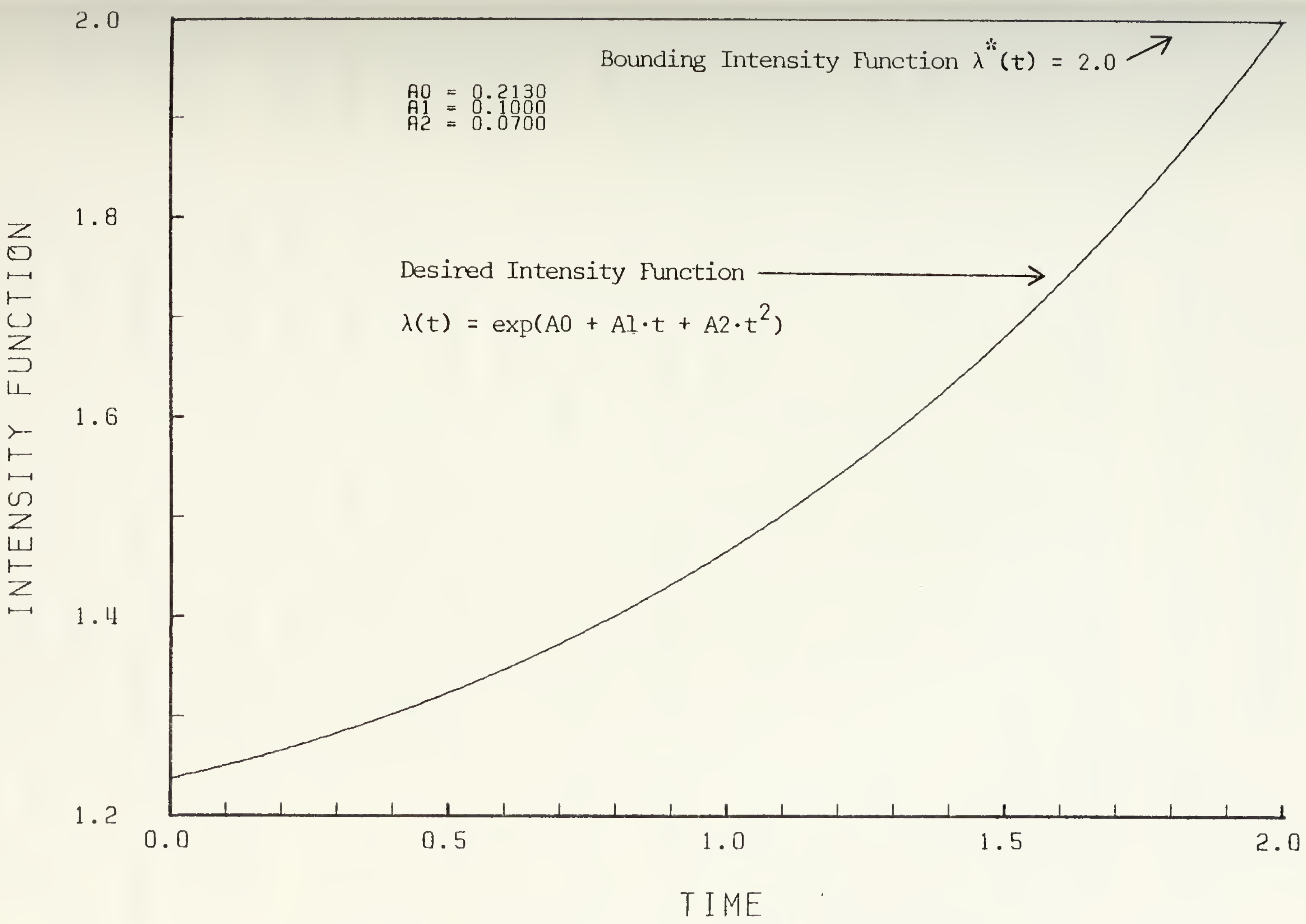

Figure 1. SAMPLE INTENSITY FUNCTIONS OF BOUNDING AND DESIRED POISSON PROCESSES FOR DEGREE-TWO EXPONENTIAL POLYNOMIAL CASE

III. ALGORITHMS CONSIDERED

A. POISSON DECOMPOSITION AND GAP-STATISTICS ALGORITHM

1. Usage

This algorithm is the one found by PATROW [Ref. 3] to be most efficient in simulation of the degree-two exponential polynomial class of intensity functions and its implementation by PATROW was confined to that group. Basically, the approach is to decompose the intensity function (which is of the form $\lambda(t) = \exp(a_0 + a_1 t + a_2 t^2)$) into a degree-one exponential polynomial function, $\lambda_L(t) = \exp(b_0 + b_1 t)$, and a difference function, $\lambda_D(t) = \lambda(t) - \lambda_L(t)$. The points or events in the process with the degree-one exponential polynomial function, $\lambda_L(t)$, are generated over the interval $(0,t']$ utilizing the computationally fast gap-statistics algorithm. The points in the process with intensity function $\lambda_D(t)$ are generated using conditioning-acceptance-rejection techniques. The two event streams are then superposed to produce the event stream for the non-homogeneous Poisson process with the intensity function $\lambda(t)$.

In the case where $\lambda(t)$ has an internal maximum or minimum in the interval $(0,t']$, the interval is partitioned and treated as two separate intervals for simulation with the event streams being merged in the final step.

Efficiency is optimized by maximizing the area under the degree-one exponential polynomial intensity

function, $\lambda_L(t)$, subject, of course, to the constraint $\lambda_L(t) \leq \lambda(t)$ for $0 \leq t \leq t'$. This maximizes the use of the faster gap-statistics algorithm and minimizes the use of the conditioning-acceptance-rejection algorithm which is slow relative to the gap-statistics algorithm.

PATROW [Ref. 3] deals extensively with the details of this algorithm and it is consequently presented here only in outline form.

2. Algorithm Statement

a. Categorize the intensity function, $\lambda(t)$, into one of six cases by examination of the coefficients a_1 and a_2 in $\lambda(t) = \exp(a_0 + a_1 t + a_2 t^2)$. Examples of each of these cases are shown in Figures 2 through Figure 7 located in Section IV.

b. (1) If $\lambda(t)$ is monotone increasing or monotone decreasing over the interval (Cases I, II, IV and V; see Figures 2,3,5 and 6), decompose $\lambda(t)$ into $\lambda_L(t)$, which is degree-one exponential polynomial, and $\lambda_D(t) = \lambda(t) - \lambda_L(t)$. Thus the decomposed functions have the forms:

$$\lambda_L(t) = \exp(b_0 + b_1 t)$$

$$\lambda_D(t) = \exp(a_0 + a_1 t + a_2 t^2) - \exp(b_0 + b_1 t)$$

and

$$\lambda(t) = \lambda_L(t) + \lambda_D(t).$$

Choose b_0 and b_1 so as to maximize the area under $\lambda_L(t)$ subject to $\lambda_L(t) \leq \lambda(t)$ for all t in $(0,t']$.

(2) If $\lambda(t)$ is not monotone over the interval (Cases III and VI; see Figures 4 and 7), partition the interval $(0,t']$ into two disjoint, contiguous subintervals, $(0,b]$ and $(b,t']$. Choose b as the (unique) point where $\lambda(b)$ is a maximum (minimum) of $\lambda(t)$ over $(0,t']$. Treat each subinterval as in b.(1), applying subsequent steps on each subinterval separately, and combining results as the final step.

c. Generate points in the Poisson process with degree-one exponential polynomial intensity function, $\lambda_L(t)$, using the gap-statistics method.

d. Generate and order points in the Poisson process with intensity function $\lambda_D(t)$ using the conditioning-acceptance-rejection method.

e. Merge (superpose) the two event streams from Step 3 and Step 4. The merged stream is from the non-homogeneous Poisson process with intensity function $\lambda(t)$.

B. THE BASIC THINNING ALGORITHM

1. Usage

The thinning theorem is implemented in a straightforward manner. Typically, the bounding process used is homogeneous Poisson with constant rate λ^*, where λ^* is an

upper bound of $\lambda(t)$ over $(0,t']$. In this case efficiency is optimized if λ^* is the least upper bound (LUB) of $\lambda(t)$ over $(0,t']$.

2. Algorithm Statement

a. Generate events in the Poisson process $\{N_t^* : t \geq 0\}$ with rate function $\lambda^*(t)$ in the fixed interval $(0,t']$. If the number of events generated, n^*, is such that $n^* = 0$, exit; there are no events in the process $\{N_t : t \geq 0\}$.

b. Denote the (ordered) events by $T_1^*, T_2^*, \ldots, T_n^{*}*$. Set $i = 1$ and $k = 0$.

c. Generate U_i, uniformly distributed between 0 and 1. If $U_i \leq \lambda(T_i^*)/\lambda^*(T_i^*)$, set k equal to $k+1$ and $T_k = T_i^*$.

d. Set i equal to $i+1$. If $i \leq n^*$, go to c.

e. Return $T_1, T_2, \ldots, T_n$, where $n = k$, and also n.

C. THE ONE-AT-A-TIME THINNING ALGORITHM

1. Usage

In some event-step simulations, it is customary or necessary to generate only one event at a time, rather than an array of events. The thinning algorithm is easily modified to generate the next event in the non-homogeneous Poisson process with intensity function $\lambda(t)$. In this case, the algorithm utilizes the time of the last event, T_{i-1}; the right hand limit, t', of the fixed interval over which the process is being simulated; and the bounding process intensity function, $\lambda^*(t)$. All variates are generated

one at a time, thus no arrays are required for storage. The output is T_i, the time of the next event, if any, in the interval $(T_{i-1},t']$.

The algorithm is stated here for the case in which the bounding process is homogeneous Poisson, i.e., $\lambda^*(t) = \lambda^*$, a constant and an upper bound of $\lambda(t)$. Specifically:

T_i is obtained by generating and cumulating exponential (mean $= 1/\lambda^*$) random variates $E^*_{i,1}$, $E^*_{i,2}$, ... , for i = 1, 2, ..., until for the first time,

$$U_{i,j} \leq \lambda(T_{i-1} + E^*_{i,1} + \dots + E^*_{i,j})/\lambda^*$$

A detailed algorithmic statement of this procedure follows.

2. Algorithm Statement

If i = 1, set $T_{i-1} = 0$ (i.e. the left end point of the interval), otherwise, T_{i-1} is known. Then for each i = 1, 2, ... , the time, T_i, of the event in the non-homogeneous Poisson process is given by the following:

a. Set j = 1

b. Generate $E^*_{i,j}$, an exponential random variate with mean $1/\lambda^*$. If $T_{i-1} + \sum_{k=1}^{j} E^*_{i,k}$ is greater than t', exit; there are no more points in the interval $(T_{i-1},t']$.

c. Generate U_j, uniformly distributed between 0 and 1. If

$$U_j \leq \lambda(T_{i-1} + \sum_{k=1}^{j} E^*_{i,k})/\lambda^*$$

set

$$T_i = T_{i-1} + \sum_{k=1}^{j} E^*_{i,k}$$

and exit.

d. Otherwise set j = j+1; go to b.

Note: $U_{i,j}$ and U_j are uniformly distributed between 0 and 1.

IV. METHODOLOGY FOR ALGORITHM COMPARISON

A. MEASURES OF EFFECTIVENESS

Two quantifiable measures of effectiveness were chosen as yardsticks for algorithm comparison. These were computational speed and computer memory requirements. Some other considerations, such as programming ease and robustness, are discussed in Section VI. It must also be recalled that the classes of intensity functions for which the two algorithms are usable are different. The Poisson decomposition and gap-statistics algorithm is only easily implemented for a restricted set of intensity functions, those of the form $\lambda(t) = \exp(a_0 + a_1 t + a_2 t^2)$, i.e. the degree-two exponential polynomial. Conversely, the thinning algorithm is valid for any positive intensity function. Thus a direct comparison can only be made in that subset of intensity functions for which both algorithm implementations are valid, the degree-two exponential polynomials.

PATROW [Ref. 3] developed six sample intensity functions, all special cases of the degree-two exponential polynomial, and these are used herein as the test cases. These are described in Section IV.C.3 below.

1. Computational Speed

Typically, computer time is a costly commodity in economic terms. It may also be a significant factor in determining more mundane considerations such as job

priority and thus turn-around time. Thus computer run time is a natural candidate as a measure of effectiveness for comparing competing algorithms.

PATROW [Ref. 3] utilized a procedure in which event streams from each of six sample intensity functions were replicated several times in "packages". The number of replications was large if the expected number of events in the event stream was small, and vice versa. Thus the product of the number of events times the number of replications was kept on the same order of magnitude. For simplicity, the same technique was used here although results showed a wide variation in the run times for the six packages.

Programming of the thinning algorithm was done so as to minimize run time while maintaining parity with the Poisson decomposition and gap-statistics algorithm wherever direct comparison could be made. For example, shuffled random numbers are called in both programs and both are dimensioned to accommodate event streams of up to 5000 events.

Undoubtedly further programming refinements exist which might increase slightly the speed of one or the other algorithm. Also, different computers might have unique features which could be exploited. The overall purpose here was to obtain a relative order of magnitude comparison and it believed that this objective was accomplished in every meaningful sense.

2. Computer Memory Requirements

This is the second obvious means of comparing two algorithms. Again, some core reduction could undoubtedly be made by a sophisticated programmer. Most notably, core requirements can be reduced substantially if only one non-homogeneous Poisson variate is generated at a time (the one-at-a-time algorithm) but this has the predictable effect of increasing execution time considerably (see Tables IV, V, and VI).

B. MEASUREMENT CONSIDERATIONS

Measurement of computer memory requirements is straightforward and deterministic.

Measurement of computational speed, more specifically Central Processing Unit (CPU) time, is quite another matter. First of all, the number of events in each replication of the non-homogeneous Poisson process varies causing CPU time to be a random variable. More important, however, are the effects of internal computer procedures.

In the first place, the so-called CPU time printed out on the normal IBM-360 output has only a general relationship to the actual computational time required by the CPU. This is caused by the addition of certain "overhead" time. This overhead time is a function of the number of other programs in the system as well as such factors as compilation and linkage times. Thus the same program run at two different times may differ in "CPU time" by a factor of two or more.

Program execution times were isolated from compilation and linkage times by the use of a system subroutine, GETIME. This subroutine allows the user to initialize an internal timer within the program and read cumulative time at various points in the program. Although this method is not exact, it does measure actual elapsed CPU time to within a small fraction of a second. This does not, however, entirely alleviate the time-of-day effect experienced when running the same program at different times. That is, although the elapsed CPU time can be measured accurately, the same program will generally have somewhat different execution times each time it is run [Ref. 4]. Theoretically, the execution times would be constant for stand-alone runs, i.e. runs with no other competing programs in the system. This is rarely realized in practice.

These considerations lead to the development and use of the side-by-side setup described below. This method appears to be statistically sound as a means of dealing with the problems of time measurement. Due to the differences in execution times noted, the best measure of effectiveness was determined to be a ratio of execution times for the respective algorithms, rather than absolute times.

C. TEST SETUP

1. Computational Speed

The central idea here was to equalize the effects of non-essential processes on each algorithm. This was accomplished by the following algorithm:

1. Set k = 1.
2. Zero internal time clock.
3. Call Algorithm A. Replicate M times.
4. Read internal time clock. Store time.
5. Zero internal time clock.
6. Call Algorithm B. Replicate M times.
7. Read internal time clock. Store time.
8. Set k = k + 1. If k is greater than k_{max}, go to 9. Otherwise go to 2.
9. Compute mean and variance of the k_{max} execution times for each algorithm.
10. Compute ratio of means.

This procedure was used in all comparisons. M, the number of replications per package, varied between 30 and 100 as discussed above. K_{max}, the number of times each package was replicated, was typically set equal to thirty.

2. Computer Memory Requirements

To measure computer memory requirements, a small main program was written, calling the subroutine which implemented the program being measured. Total program length in bytes was obtained from the standard computer output and the core alloted to the main program was subtracted to obtain the desired figure. This includes all library routines and arrays for storage of event times and arrays of random variates.

The core requirements are deterministic in that they do not change from one run to another but are strictly a function of the program coding.

3. Test Cases Utilized

PATROW [Ref. 3] developed six sample intensity functions representing the possible variations in sign and relative magnitude of the coefficients a_1 and a_2 in the exponential polynomial, $\exp(a_0 + a_1 t + a_2 t^2)$.

Since the sample intensity functions were designed to test different aspects of the Poisson decomposition and gap-statistics algorithm, they were also used for comparison here. Although each algorithm is affected by different considerations, the test cases do, coincidentally, put the thinning algorithm through its paces.

For continuity, the six test cases, or sample intensity functions, are presented below in Figures 2 through 7.

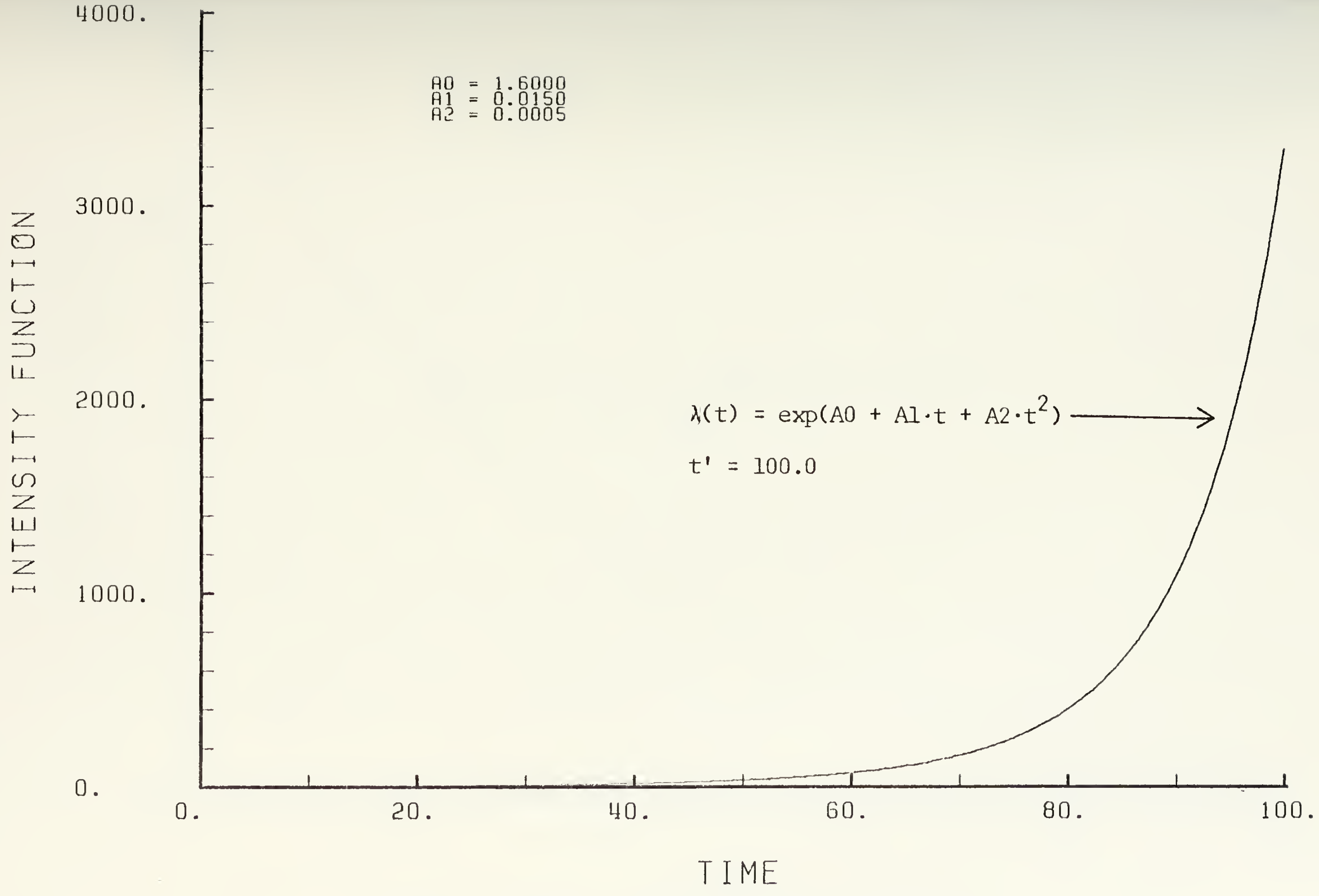

Figure 2. CASE I - SAMPLE INTENSITY FUNCTION

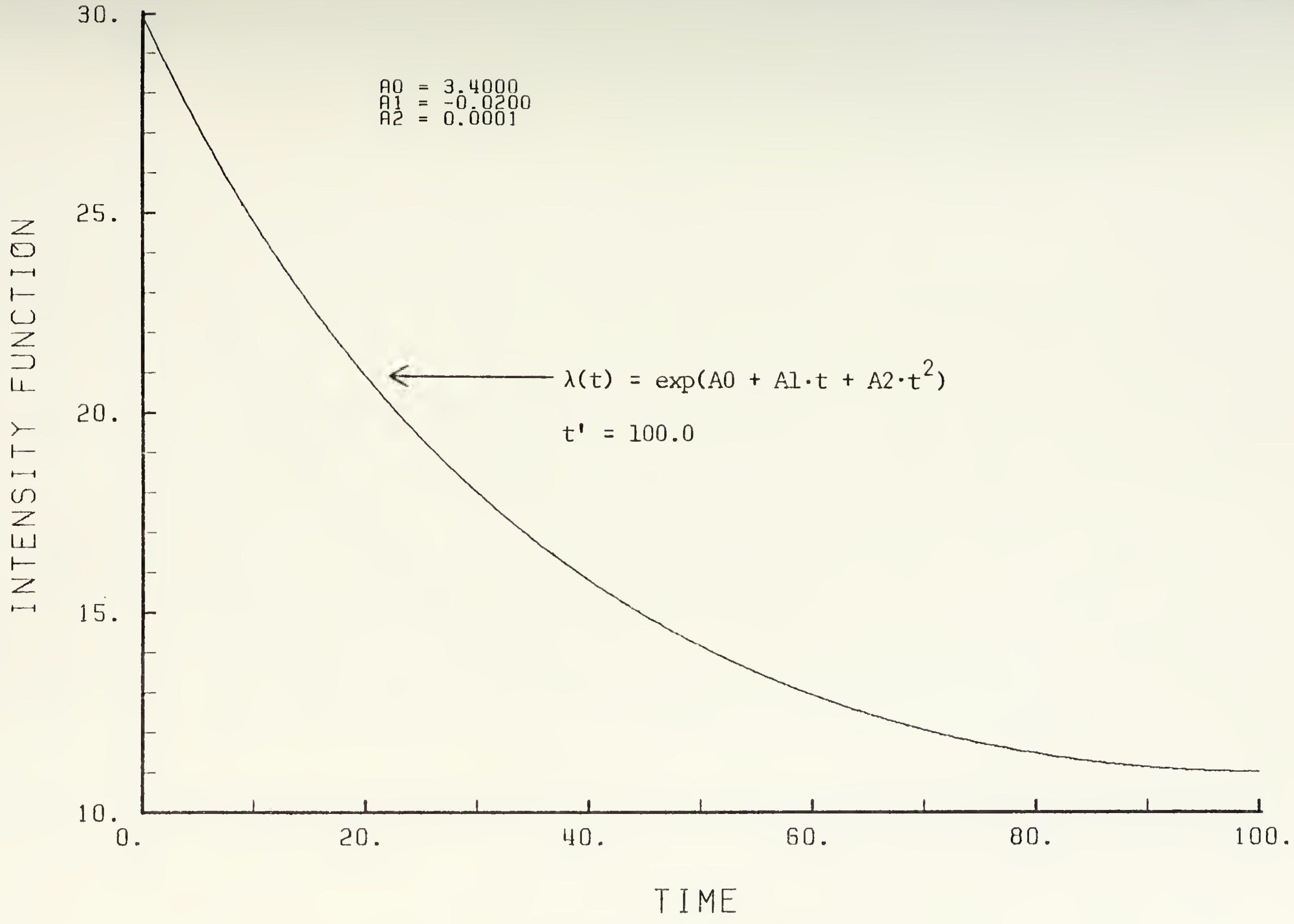

Figure 3. CASE II - SAMPLE INTENSITY FUNCTION

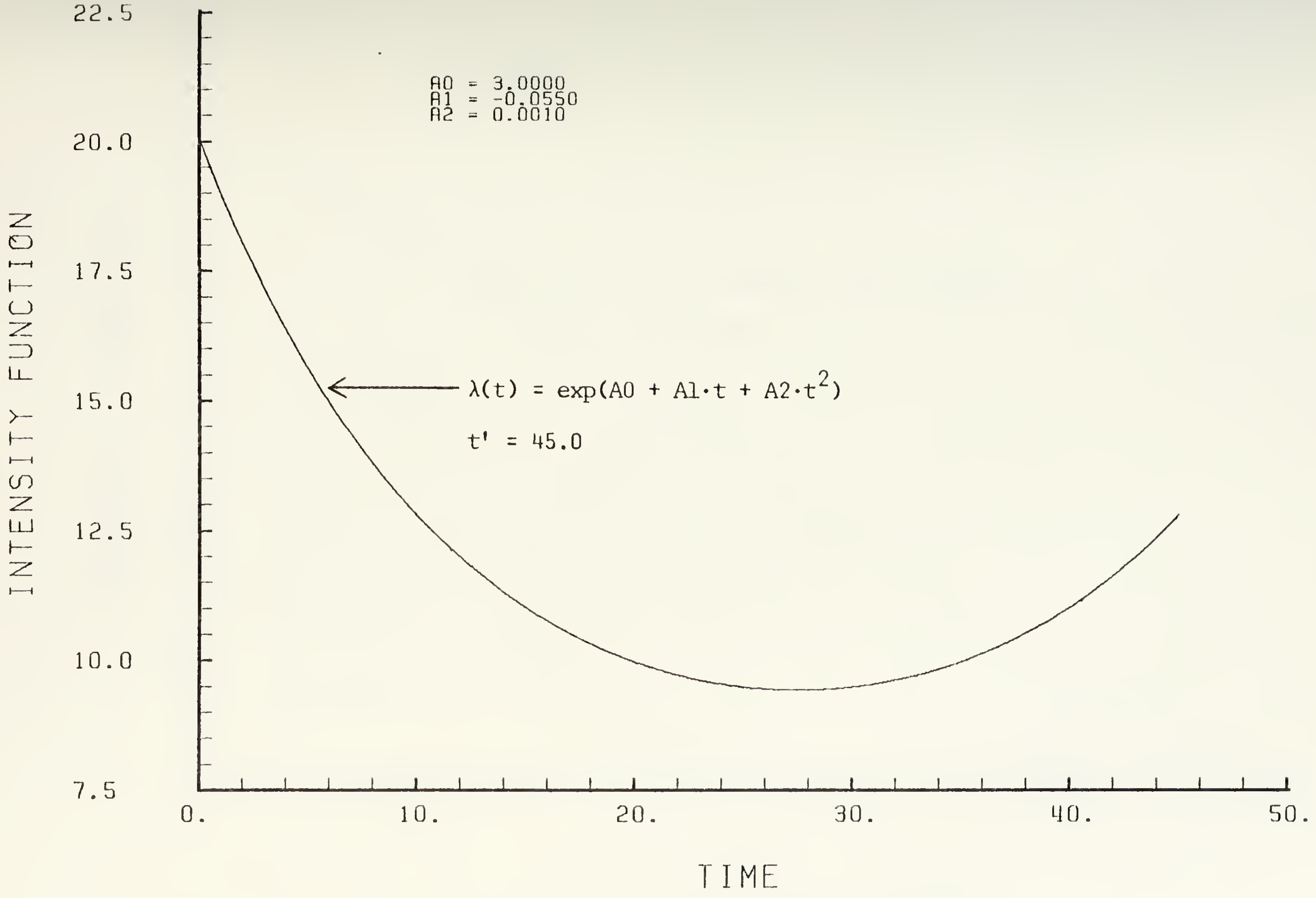

Figure 4. CASE III - SAMPLE INTENSITY FUNCTION

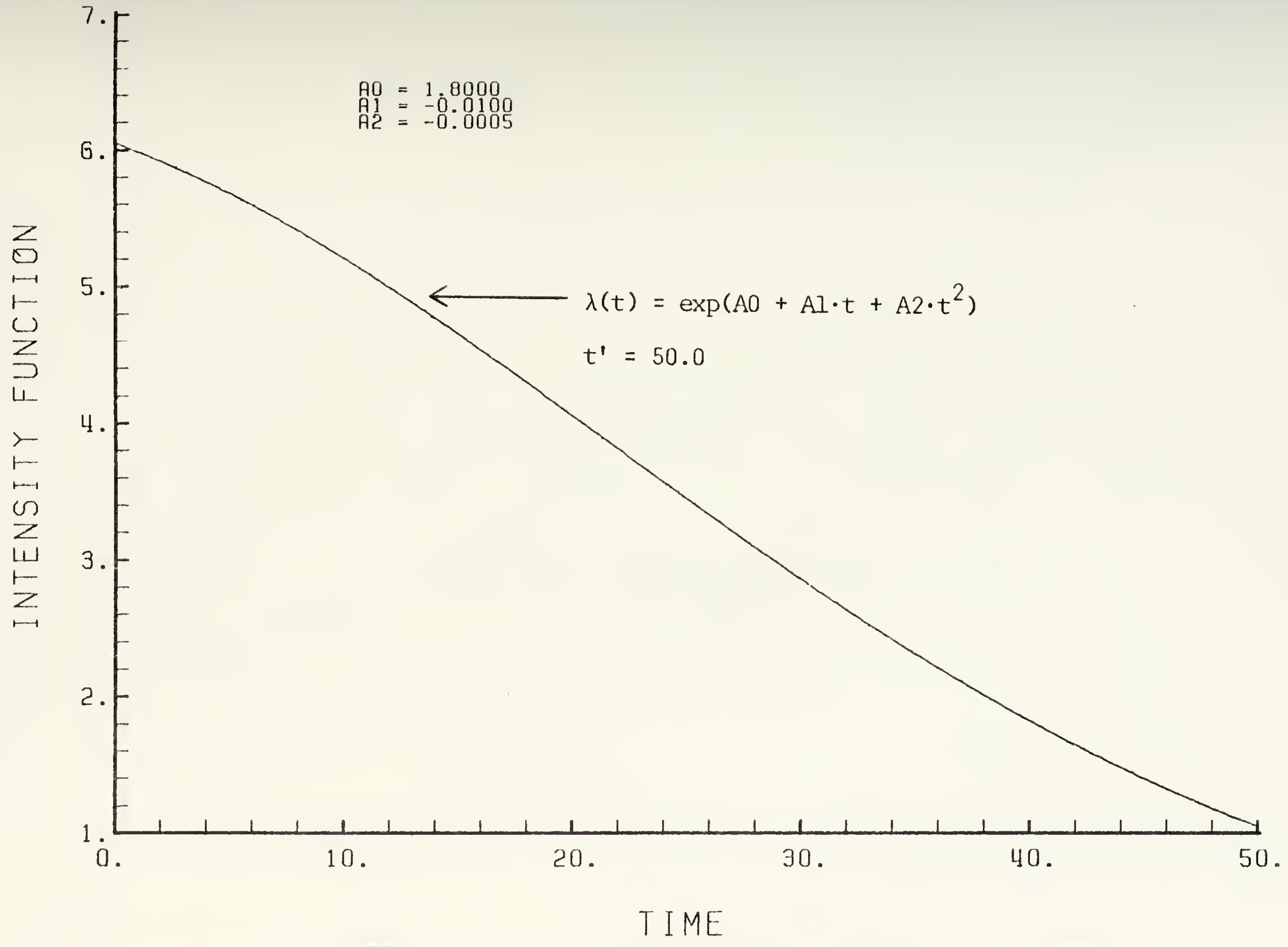

Figure 5. CASE IV - SAMPLE INTENSITY FUNCTION

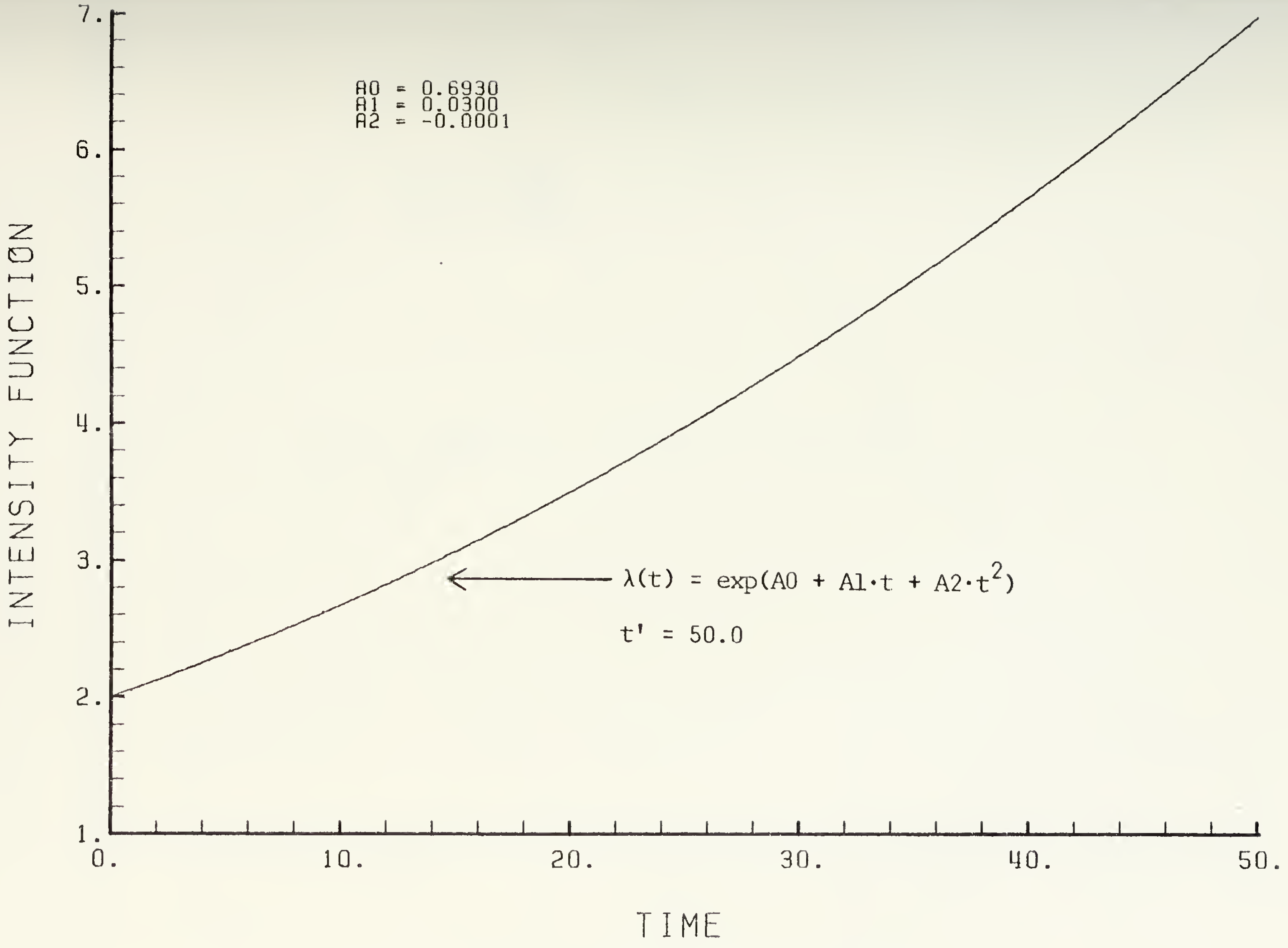

Figure 6. CASE V - SAMPLE INTENSITY FUNCTION

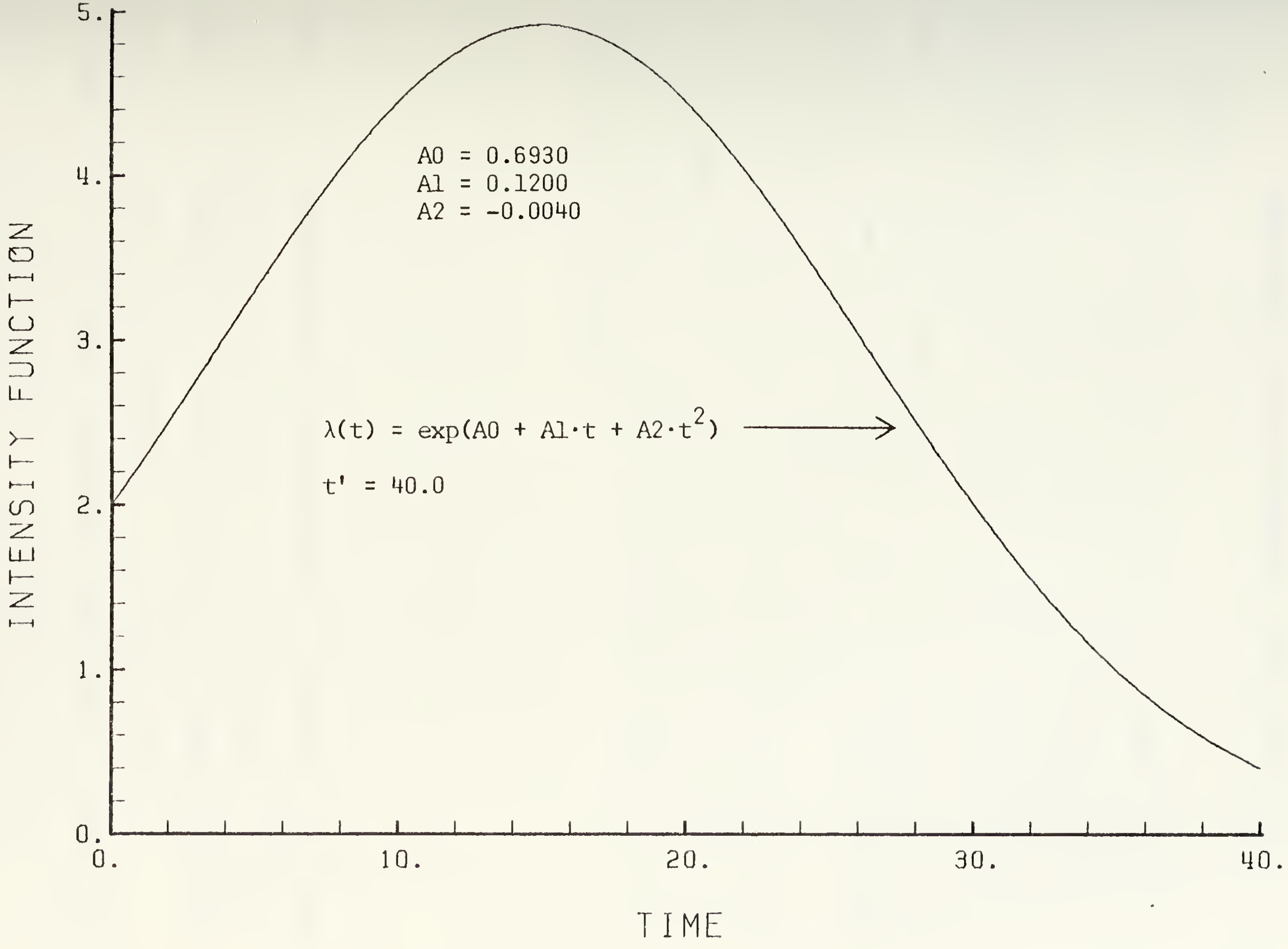

Figure 7. CASE VI - SAMPLE INTENSITY FUNCTION

V. EFFICIENCY AND PROGRAMMING CONSIDERATIONS

A. GENERAL

This section deals with factors which affect the performance of the thinning algorithm. Four specific areas are presented in which significant gains in terms of computational speed may be realized. With the exception of Section V.D (which applies only to the case of exponential polynomial intensity functions) these considerations apply to the general class of intensity functions.

In general application, one of the primary indicators of efficiency is the relative size of the area under the intensity function to that of the bounding function, i.e. the ratio, R, given by:

$$R = \int_0^{t'} \lambda(t)dt / \int_0^{t'} \lambda^*(t)dt$$

Since both numerator and denominator are simply the respective integrated rate functions, $\Lambda(t)$, evaluated at either end of the interval, R is the ratio of the expected number of events in the two processes, i.e. $E[N_{t'}]/E[N^*_{t'}]$.

Case 1 of the sample intensity functions is particularly illustrative (see Figure 2). The intensity function $\lambda(t) = \exp(1.6 + 0.015t + 0.0005t^2)$ is bounded on the

interval (0,100] by a least upper bound (LUB) of 3294.47. If a homogeneous Poisson process with rate equal to the LUB is used as the bounding function, $E[N^*_{t'}] = 329{,}447$ points will be generated on the average. Of these, all but 1464 will be rejected on the average (i.e. $E[N_{t'}]$). The ratio of the respective expected values is thus $1464/329{,}447 = 0.0044$ = the ratio of the areas under the intensity functions.

Thus a rough relative measure of the efficiency of the thinning process in a particular situation can be gained by examining a graph of the two intensity functions, even if the expected values are not easily calculated. This procedure may also be an indicator in deciding whether to partition the interval and use different bounding functions on each subinterval.

B. UTILIZATION OF ARRAYS OF RANDOM VARIATES

Computer generated random variates are used both in generating the points of the bounding process $\{N^*_t : t \geq 0\}$ and in the actual thinning process itself. Since the number of variates required is typically large, efficient generation becomes a programming consideration for medium to large scale simulations.

The basic thinning program presented herein requires both exponential and uniform random variates. Both types are obtained utilizing the random number package, LLRANDOM, developed by LEARMONTH and LEWIS [Ref. 5]. Shuffled

exponential random variates of mean 1.0 are generated using the SEXPON subroutine while shuffled uniform (0,1) random variates are obtained from the SRAND subroutine. Both of these routines offer considerable "economies of scale" in terms of time when multiple numbers or arrays of variates are generated at once, as opposed to one-at-a-time generation. Using the test setup of Section IV.C, average times to generate varying quantities of random numbers were determined. Table I reveals the relative savings realized by calling large arrays of random numbers. Thus considerable time can be saved by generating all required random numbers from one subroutine call. Programming difficulties involve deciding how many variates to generate. The general goal is to generate as many as needed while keeping the unused excess to a minimum. The balance used was to generate the expected number required plus an excess of four standard deviations. For example, in the generation of the bounding process, the expected number of points, $E[N^*_{t'}]$ is $\lambda^* \cdot t'$ and the variance is the same. Thus the number of exponentials called was $y + 4\sqrt{y}$ where $y = \lambda^* \cdot t'$. Provision is made for the unlikely (1 in 40,000) case that more are required.

For specific applications this procedure could be improved slightly. For example, if the expected number of points, $E[N^*_{t'}]$, is small, e.g. 100, then the expected excess (four standard deviations) comprises forty percent

Type of Variate	Number Called	Total Time (μsec)	Mean Time Per Variate (μsec)
Exponential	1	784	784
Exponential	10	1293	129
Exponential	100	7343	73
Exponential	1000	68046	68
Uniform	1	1213	1213
Uniform	10	1381	138
Uniform	100	3276	33
Uniform	1000	21544	22

Sample Size = 200 (each grouping)

Table I

Generation Times for Arrays of Shuffled Random Variates From LLRANDOM

of the total whereas for large $E[N_{t'}^{*}]$, e.g. 4000, the expected "waste" is only about six percent. In the former case, reducing the "padding" to one or two standard deviations would, on the average, increase efficiency slightly although the probability of a second subroutine call for more random variates would increase.

As an example, if we were to call 100 exponential variates one at a time, the total time is 78,400 μsec, compared to 7,343 μsec for 100 exponential variates called in an array. For $100 + 4\sqrt{100} = 140$ variates, the time is still small compared to 78,400 μsec.

C. UTILIZATION OF INTENSITY FUNCTION LOWER BOUND

One of the most time consuming repetitive operations is the computation of the intensity function value, $\lambda(t)$, during the thinning process. In the case of the exponential polynomial intensity function, this involves one power, two multiplications, two additions, and one exponentiation for each point generated in the bounding process. Since points are accepted for the non-homogeneous Poisson process whenever the uniform (0,1) thinning variate is less than the ratio $\lambda(t)/\lambda^*(t)$, considerable time savings result if the intensity function has a positive lower bound, say $\underline{\lambda}$, since points are always accepted when the uniform (0,1) variate is less than the ratio $\underline{\lambda}/\lambda^*$. In the general case, this ratio must be calculated only once. The expected number of intensity function computations which are alleviated by the use of the lower bound is given by $(\underline{\lambda}/\lambda^*)E[N^*_{t'}]$ where $\underline{\lambda}$ is a lower bound of the intensity function; λ^* is an upper bound of the intensity function (both bounds are over the interval (0,t']) and $E[N^*_{t'}]$ is the average number of points to be thinned, i.e. the average number of events in the bounding process.

It is clear that the closer $\underline{\lambda}$ is to being the greatest lower bound (GLB) and the closer λ^* is to being the LUB, the more efficient the program.

If the intensity function is strictly non-decreasing a further (and potentially great) improvement is realized by initially setting $\underline{\lambda}$ equal to $\underline{\lambda}$, and then setting it

subsequently equal to the last value of the intensity function, $\lambda(t)$. This results in a monotone increasing lower bound and thus a decreasing probability of evaluating the intensity function.

Test cases II through VI were run side by side with and without the use of a lower bound for the intensity function. On the average, the program which did not utilize a lower bound required twenty percent more time than the program using a lower bound. Please see Appendix B for case-by-case comparison.

D. UTILIZATION OF EXPONENTIAL VARIATES FOR THINNING OF EXPONENTIAL POLYNOMIAL INTENSITY FUNCTIONS

The time requirements for evaluating $\lambda(t)$ were discussed in Section V.C above. In the case of exponential polynomial intensity functions, e.g. $\lambda(t) = \exp(a_0 + a_1 t + a_2 t^2)$, the major contributor to computation time is the exponentiation operation. Exponentiation can be avoided by utilizing the following relationship:

$$U_i \le \lambda(t)/\lambda^* \quad \text{if and only if}$$

$$E_i = -\ln U_i \ge \ln\lambda^* - \ln\lambda(t) = \ln\lambda^* - (a_0 + a_1 t + a_2 t^2),$$

where:

U_i is a uniform (0,1) random variate,

E_i is an exponential random variate with mean one.

Thus the thinning test to accept points from the bounding process becomes:

If $E_i \geq \ln\lambda^* - (a_0 + a_1 t + a_2 t^2)$, for $t = T_i^*$, accept T_i^* as a point in the non-homogeneous Poisson process with rate $\lambda(t)$; otherwise, reject T_i^* (i.e. thin it).

The key to this relationship lies in the fact that if U is distributed uniform (0,1), then -ln U is distributed as a unit exponential variate, i.e. an exponential variate with mean one. This is shown by the following:

Let U be uniform (0,1).

Then $P\{U \leq x\} \equiv P\{\ln U \leq \ln x\} \equiv P\{-\ln U \geq -\ln x\}$

but $P\{U \leq x\} = x$, thus let $y = -\ln x$,

then $P\{-\ln U \geq y\} = \exp(-y)$.

Thus -ln U is distributed as a unit exponential variate.

Although more time is required to generate the exponential random variates for thinning than the uniforms, the alleviation of the exponentiation operation more than compensates for the additional generation time. This is because SEXPON, the portion of LLRANDOM which generates exponentials, generates exponential variates by the Marsaglia "rectangle-wedge-triangle" method, which is faster than taking logarithms.

Since exponential random variates are used in the generation of the bounding homogeneous Poisson process, an additional time savings can be realized by using the variates

which are left over (i.e. not used) from generating the bounding process (these are generated in arrays).

For the test cases considered, use of exponentials for thinning resulted in an average time savings of ten percent. Please see Appendix B for case-by-case results.

E. RECYCLING OF THINNING VARIATES

As mentioned above, a uniform or exponential random variate is required for each point to be "thinned". Each of these variates requires a significant amount of time for generation. Obviously a time savings would be realized if fewer variates were required.

1. Recycling of Uniform Variates

Assume U_i is uniform (0,1) but that its value is unknown. Assume then that further information becomes available that U_i is less than a ($0 \leq a \leq 1$), but its value is still unknown. Then U_i is uniformly distributed over the interval (0,a). If U_{i+1} is then computed by "scaling up" U_i, i.e. dividing U_i by a, then U_{i+1} is uniform (0,1). Similarly, if U_i is uniform (0,1) and subsequent information places it somewhere above a, then $U_{i+1} = (U_i - a)/(1 - a)$ is uniform (0,1). Thus by conditioning on whether the variate is greater than or less than a given value, a new variate can be computed with the desired properties. Moreover, this variate is independent of its predecessor.

In the thinning algorithm, each point is tested using a uniform (0,1) variate. Specifically, if $U_i \leq \lambda(T_i^*)/\lambda^*$

the point T_i^* is accepted as a point in the non-homogeneous Poisson process. Since the ratio $\lambda(T_i^*)/\lambda^*$ is between zero and one, and the only test is whether U_i is less than or greater than the ratio $\lambda(T_i^*)/\lambda^*$, the next uniform (0,1) variate, U_{i+1}, can be generated using the rules above. The algorithm is:

1. Let U_i be uniform (0,1). If U_i is less than $a = \lambda(T_i^*)/\lambda^*$, let $U_{i+1} = U_i/(\lambda(T_i^*)/\lambda^*)$; exit.
2. Otherwise let $U_{i+1} = [U_i - (\lambda(T_i^*)/\lambda^*]/[1-(\lambda(T_i^*)/\lambda^*)]$

U_{i+1} is uniform (0,1).

In theory, only one uniform random variate is required for the entire thinning process! In computational practice, however, care must be exercised because of the finite capacity of the computer to represent numbers. After ten to twenty divisions the scaled uniform number will consist only of low-order bits of the random number and these are usually not uniformly distributed.

If the intensity function has a positive lower bound, further efficiencies can be gained, in combination with the procedures of Section V.C above. Since multiplication is computationally faster than division, the value $1/(\underline{\lambda}/\lambda^*) = \lambda^*/\underline{\lambda}$ can be precomputed and stored. Thus if $U_i \leq \underline{\lambda}/\lambda^*$, $U_{i+1} = U_i \cdot \lambda^*/\underline{\lambda}$ can be computed as the next thinning variate. Note that no intensity function calculation is required. However, if $U_i > \underline{\lambda}/\lambda^*$, the thinning method proceeds with the next step, evaluating the intensity function at the point T_i^* and determining whether or not to thin the point.

Now, further information is known about U_i. Specifically, either $U_i > \lambda(T_i^*)/\lambda^*$, in which case T_i^* is thinned, or $\underline{\lambda}/\lambda^* < U_i \leq \lambda(T_i^*)/\lambda^*$. In either case U_{i+1} can be computed by "scaling up" U_i.

Thus, the algorithm for recycling uniform random variates for thinning is as follows:

1. If $U_i \leq \underline{\lambda}/\lambda^*$, let $U_{i+1} = U_i \cdot \lambda^*/\underline{\lambda}$ and exit.
2. If $\underline{\lambda}/\lambda^* < U_i \leq \lambda(T_i^*)/\lambda^*$, let $U_{i+1} = (\lambda^* \cdot U_i - \underline{\lambda})/(\lambda(T_i^*) - \underline{\lambda})$ and exit.
3. Otherwise, $U_i > \lambda(T_i^*)/\lambda^*$, let $U_{i+1} = (\lambda^* \cdot U_i - \lambda(T_i^*))/(\lambda^* - \lambda(T_i^*))$.

By precomputing $\lambda^*/\underline{\lambda}$, this recycling procedure requires only one multiplication in the case where $U_i \leq \underline{\lambda}/\lambda^*$. Otherwise one multiplication, two subtractions and one division are required. In either case the recycling procedure is generally faster than generating uniform random variates from a random number generator, even when a logical IF statement is added to check for extreme values ("small bits").

2. Recycling Of Exponential Variates

This section applies only where the intensity function is exponential polynomial. Here the possibilities are less promising. In the general case where no lower bound, $\underline{\lambda}$ is used, the following algorithm would apply:

If $E_i \geq \ln \lambda^* - \ln \lambda(T_i^*)$, let $E_{i+1} = E_i - \ln \lambda^* + \ln \lambda(T_i^*)$

Otherwise

Let $E_{i+1} = \ln (\lambda^* - \lambda(T_i^*))/(\lambda^* \cdot \exp(-E_i) - \lambda(T_i^*))$

where E_j is a unit exponential random variate.

In the first case, $E_i \geq \ln \lambda^* - \ln \lambda(T_i^*)$, a time savings would generally be realized since $\ln \lambda^*$ could be computed once and stored and $\ln \lambda(T_i^*)$ is simply the value of the polynomial, i.e. $a_0 + a_1 T_i^* + a_2 T_i^{*2}$, which must be computed in any event. In the second case, however, the cure is truly worse than the illness. It is faster to simply generate another exponential variate, assuming they are called in arrays.

For there to be a time savings, however, it must be possible to make a reasonable prediction of the number of exponentials which must be generated. Otherwise an excessive number of calls to the random number generation subroutine may destroy the gains made through recycling.

In the case where a lower bound, $\underline{\lambda}$, for the intensity function exists and is positive, it is possible to determine the expected number of exponentials which must be generated. Variates are reused if they are greater than $\ln(\lambda^*/\underline{\lambda})$. That is, if $E_i > \ln(\lambda^*/\underline{\lambda})$, then $E_{i+1} = E_i - \ln(\lambda^*/\underline{\lambda})$. Otherwise, a new (i.e. non-recycled) variate is used. Thus the probability of not recycling, p, is $\underline{\lambda}/\lambda^*$ and the number of variates required is binomial with mean $n^* p$, where n^* is the number of points to be thinned.

Empirical results for the five test cases considered are shown in Appendix B. Using the calling rule of expected number plus four standard deviations for generating thinning exponentials yielded inconclusive results as compared with

the procedure in which exponential thinning variates are generated in arrays with no recycling. As expected, for larger N_t, (Cases II, III and V), recycling provided a slight time advantage (seven percentage maximum) while for small N_t, (Cases IV and VI) recycling was slower. In case VI, recycling caused run time to be approximately five percentage greater than that without recycling. Using a calling rule of expected number plus two standard deviations reduced the disadvantage slightly to four percent. The reason that recycling can cause longer run times than not recycling is that an additional logical IF statement is required for the recycling program. Again, when exponentials are used for thinning and the mean number of points to be thinned is on the order of two or three hundred, it is probably not worth the effort to recycle. If several thousand "thinnings" are required, the savings may indeed be worthwhile.

Results were somewhat surprising in the general class of intensity functions for which uniform variates are used for thinning. It was expected that a significant savings would be realized since the uniform variates can be recycled in all situations. In fact, test runs were run in which only one uniform variate was generated for the entire run with recycling used throughout. The only program statement which added time was a logical IF statement to preclude dividing by zero and to diminish the probability of small bit usage. As expected, some bias was experienced in the mean and an unusually high variance was noted, indicating a

low degree of "fidelity" to the true non-homogeneous Poisson process being simulated. Of particular interest however, were the results on execution time. Since this setup essentially gives the lower bound on execution time for recycling of uniform variates. it was expected that significant time savings would be realized in comparison to no recycling. In practice, however, the savings were minimal, with a maximum of only three percent savings. This is attributed to the efficiency of the LLRANDOM package in generating uniform variates with the logical IF statement being only a secondary cause to longer execution time.

Appendix B shows results for both exponential and uniform thinning variates.

The key point to keep in mind here is that the above results reflect the case where thinning variates are called in arrays, i.e. many at a time. Thus the comparison is between a very "fast" variate and recycling. As discussed above, calling by arrays results in considerable time savings compared to one-at-a-time generation (up to fifty times faster). Thus, in the case where a slower random number generator is utilized or where variates are called one at a time, use of the lower bound may indeed result in considerable time savings.

F. FINAL PROGRAM

A general program was developed which incorporated the efficiency considerations discussed above. The program is

general in that it can be used with any of the general class of intensity functions, whether exponential polynomial or not. The program is essentially four programs, each used in a specific case. The program classifies simulations into the four classes by asking two questions:

1. Is the intensity function exponential polynomial?
2. Does the intensity function have a positive lower bound?

The first of these determines whether uniform variates (general case) or exponential variates (exponential polynomial case) are used for thinning. The second consideration merely deletes an unnecessary logical IF statement in the case where no lower bound is used.

The computer program, NHPP, is listed after the appendices, and requires a user supplied subprogram FUNCTION FCN(T) to compute the intensity function values for each value of t. If the intensity function is exponential polynomial, only the exponent portion should be calculated, i.e. the statement $FCN = a_0 + a_1 t + a_2 t^2$ (for degree-two polynomial) should appear in the subprogram. Otherwise the entire intensity function should be evaluated.

Empirical results showed that the final program, utilizing the efficiency considerations mentioned in this chapter, resulted in a program which ran in two-thirds the time of the basic thinning program.

Please see Appendix B for case-by-case results.

VI. RESULTS, CONCLUSIONS AND RECOMMENDATIONS

A. GENERAL

This section presents the results of comparison of the Poisson decomposition and gap-statistics algorithm with three variations of the thinning algorithm. These variations are the basic thinning algorithm, the modified thinning algorithm (final program) and the special case one-at-a-time algorithm. Section B presents the performance of each of the algorithms when measured by the two measures of effectiveness, computational speed and computer memory requirements. Section C examines the results with a view toward identifying the strong and weak points of each algorithm. Section D recommends further avenues of study.

Again, in comparing the two classes of algorithms, one basic distinction must be kept in mind. That is that the Poisson decomposition and gap-statistics algorithm as implemented by PATROW [Ref. 3] is limited to a special class of intensity functions, i.e. exponential polynomial of degree two (or less). Although the algorithm could be adapted to higher order polynomials (by further bisection of intervals), the already complex programming considerations would grow significantly. In contrast, the thinning algorithm is a completely general method which is analytically valid for any functional form of permissible intensity functions (positive and right continuous).

The results presented here are necessarily limited to that class of intensity functions for which both algorithms can be compared, i.e. the degree-two exponential polynomial class. The purpose was to determine the relative performance of the thinning algorithm on this piece of common turf with the heretofore champion, Poisson decomposition and gap-statistics.

The basic result is that the thinning algorithm is indeed quite competitive with the Poisson decomposition and gap-statistics algorithm in the area of mutual validity. This, combined with its ease of programming and ability to generate variates from any intensity function, make the thinning algorithm a highly attractive tool for generating non-homogeneous Poisson processes of any type.

One shortcoming of the thinning program was revealed by the first test case considered (see Section IV.C). This is a fast rising exponential polynomial which rises from a value of five to almost 3300 over the interval (0,100]. For $\lambda^* = 3294.47$, the expected number of points in the bounding process is 329,447 while the non-homogeneous Poisson process being simulated has an expected number of only 1464 points. Thus all but one point in about 200 are thinned out. The thinning algorithm could be more efficiently adapted to this case by partitioning the interval, alleviating the necessity to store over 300,000 bounding process points. However, the efficiency involved would still be low, and the

best solution appears to be to utilize the Poisson decomposition and gap-statistics approach. The key point here is that the problem is easily recognized beforehand, as discussed in Section V.A, and avoidable.

Table II presents a general comparison of the two algorithms.

B. MEASURES OF EFFECTIVENESS RESULTS

Chapter four details the comparison procedure utilized to develop the following results.

1. The Basic Thinning Algorithm

Salient features for this case include the use of uniform variates for thinning and one-at-a-time generation of exponential and uniform variates.

Table III presents the results for each of the five test cases run. Algorithm A is computer program DEGTWO, the Poisson decomposition and gap-statistics program developed by PATROW and listed at the end of this paper. Algorithm B is computer program NHPTHN, the basic thinning algorithm, also listed.

The thinning algorithm was fastest in two out of the five test cases run and required eighty percent of the core space required for the gap-statistics algorithm.

Table VII lists core storage requirements for each algorithm.

2. The Modified Thinning Algorithm (Final Program)

This section compares the best case performance for both algorithms.

AREA	THINNING ALGORITHM IMPLEMENTATION	POISSON DECOMPOSITION AND GAP-STATISTICS ALGORITHM IMPLEMENTED BY PATROW [REF. 3]
1. Valid intensity functions	all	degree-two exponential polynomial
2. Programming complexity	simple	complex
3. Storage requirements	a. fastest algorithm - moderate	moderate
	b. one-at-a-time algorithm - small	
4. Computational speed		
a. log quadratic intensity function	wins 3 out of 6	wins 3 out of 6
b. log linear intensity function	slow	fast (2-5 times faster than thinning)

TABLE II

COMPARISON OF TWO ALGORITHMS

Case	E(N)	Algorithm	Replications Per Package	$\overline{N}$	$\hat{\sigma}^2$	Time(Sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	1610	1546	19.7272	0.5162
		B	30	1613	1734	38.2158	
III	526	A	75	526	543	17.5183	0.6050
		B	75	526	557	28.9547	
IV	174	A	100	175	176	17.2034	1.1858
		B	100	175	174	14.5081	
V	208	A	100	207	207	18.0211	1.0636
		B	100	208	204	16.9440	
VI	124	A	100	124	124	13.2313	1.3609
		B	100	124	122	9.7222	

Note: Case I not run due to thinning algorithm inefficiency (see Section IV.C.)

Algorithm A: Poisson Decomposition and Gap-Statistics (Program DEGTWO)
Algorithm B: Basic Thinning Algorithm (Program NHPTHN)

TABLE III

COMPARISON OF POISSON DECOMPOSITION AND GAP-STATISTICS AND BASIC THINNING ALGORITHMS

The modified thinning algorithm includes:

1. Use of exponentials for thinning of exponential polynomial intensity functions
2. Use of lower bounds
3. Partial recycling of exponential thinning variates
4. Use of exponential variates left over from generation of the bounding process.

Each of these refinements are discussed in Section V.

The Poisson decomposition and gap-statistics algorithm used was again the implementation by PATROW [Ref. 3], program DEGTWO. In addition to the normal running of the program, a second set of comparisons was made utilizing separately calculated values for c^*, the bound for the conditioning-acceptance-rejection routine. This is discussed by PATROW [Ref. 3]. These runs are indicated by asterisk (*) in Table IV.

In the first case, the thinning algorithm was faster in four out of the five test cases, with the best relative performance occurring in Case VI.

When the improved values of c^* were incorporated for Cases IV, V, and VI, the Poisson decomposition and gap-statistics algorithm improved substantially, winning in three out of the five test cases.

The relative advantage in computational speed was less than a factor of two in all cases with a maximum of 1.83 to 1 in favor of the thinning algorithm in case VI.

Case	E(N)	Algorithm	Replications Per Package	$\overline{N}$	$\hat{\sigma}^2$	Time(Sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	1612	1546	18.8917	.8127
		B	30	1612	1562	23.2458	
III	526	A	75	526	543	18.6622	1.0137
		B	75	524	485	18.4107	
IV	174	A	100	175	164	18.3944 (8.1610)*	1.6606 (.8548)*
		B	100	175	153	11.0771 (9.5474)*	
V	208	A	100	208	208	19.0798 (6.8877)*	1.5988 (.6734)*
		B	100	208	191	11.9336 (10.2284)*	
VI	124	A	100	124	134	15.0207 (8.2251)*	1.8299 (1.2221)*
		B	100	124	127	8.2083 (6.7303)*	

Note: Case I not run due to thinning algorithm inefficiency (see Section IV.C.)

Algorithm A: Poisson Decomposition and Gap-Statistics (Program DEGTWO)
Algorithm B: Modified Thinning Algorithm (Final Program) (Program NHPP)

* Second Run with Better c^* Value for DEGTWO (Determined Graphically)

TABLE IV

COMPARISON OF BEST CASE FOR BOTH ALGORITHMS

Core storage for the thinning algorithm was determined by using only the part of the algorithm which used exponential thinning variates. This precluded the requirement for storing 5000 uniform variates which are not used. The Poisson decomposition and gap-statistics program (DEGTWO) required about 88,000 bytes of core storage as compared to about 94,000 for the thinning program (NHPP modified to exclude unused uniform variate array). Detailed results are listed in Tables IV and VI.

3. The One-At-A-Time Thinning Algorithm

As discussed in Section II.C, the one-at-a-time algorithm was developed only to test the relative efficiency of the algorithm used to generate the next event in a non-homogeneous Poisson process. This latter requirement may arise in event-step simulation where only the next event in a non-homogeneous Poisson process is desired rather than an array containing all events in a specified interval.

Computationally, the one-at-a-time algorithm is quite similar to the basic thinning algorithm. The only essential difference is that the basic thinning algorithm generates and stores all the points in the bounding process (intensity function λ^*) before thinning, whereas the one-at-a-time algorithm generates a point in the bounding process and thins that point before continuing. The latter method removes the requirement for an additional array to store the bounding process points. This in turn saves about

20,000 bytes of core storage requirement when the programs are dimensioned to accept 5000 points.

As implemented here, the one-at-a-time algorithm simply generates the next point in the non-homogeneous Poisson process and stores it, stopping when the last point generated lies outside the interval. All variates in this program are generated one at a time. The results shown in Table V are thus a good indicator of the relative efficiency of using this method. As can be seen, the one-at-a-time algorithm (program NHPOAT) is faster than the Poisson decomposition and gap-statistics algorithm (program DEGTWO) in three of the five test cases run. This is true despite the fact that DEGTWO generates all variates in arrays, taking advantage of the time economies of scale mentioned in Section V. The one-at-a-time algorithm also requires forty percent less core space.

From the tables one can also see that since both the best case thinning algorithm (Table IV) and the one-at-a-time thinning algorithm (Table V) are compared to the Poisson decomposition and gap-statistics algorithm, it is possible to obtain a reasonable comparison of the best case thinning algorithm and the one-at-a-time thinning algorithm. For example, for the sample intensity function used in Case II (see Figure 3), the ratio (.8127/.5541) = 1.47 indicates that execution time for the one-at-a-time thinning algorithm is almost fifty percent greater than that of the

Case	E(N)	Algorithm	Replications Per Package	$\overline{N}$	$\hat{\sigma}^2$	Time(Sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	1610	1546	21.8808	.5541
		B	30	1614	1692	39.4873	
III	526	A	75	526	531	18.8685	.6274
		B	75	526	545	30.0744	
IV	174	A	100	175	172	16.0667	1.3109
		B	100	175	181	12.2567	
V	208	A	100	208	215	18.7335	1.1696
		B	100	208	199	16.0167	
VI	124	A	100	124	127	14.1217	1.510
		B	100	124	120	9.3550	

Note: Case I not run due to thinning algorithm inefficiency (see Section IV.C.)

Algorithm A: Poisson Decomposition and Gap-Statistics (Program DEGTWO)
Algorithm B: One-At-A-Time Thinning Algorithm (Program NHPOAT)

TABLE V

COMPARISON OF POISSON DECOMPOSITION AND GAP-STATISTICS AND ONE-AT-A-TIME THINNING ALGORITHM

best case thinning algorithm. The ratios for Cases III, IV, V and VI are 1.62, 1.27, 1.37, and 1.21 respectively.

The tables also demonstrate the time-of-day effect discussed in Section IV. That is, the execution time for a given program is not the same each time it is run. For example, program DEGTWO took 21.8808 seconds for the run recorded in Table V compared to 18.8917 seconds for the run recorded in Table IV. For this reason, ratios of execution times were chosen as the measure of effectiveness rather than absolute times.

A FORTRAN program listing, NHPNXT, is provided at the end of this thesis. This program generates the next point in a non-homogeneous Poisson process with a user supplied intensity subprogram, FUNCTION FCN. This program can be used in conjunction with event-step simulation programs, including SIMSCRIPT, where it is desired to minimize core space (at the expense of speed) or where only one event is desired at a time. Core requirements are shown in Table VI.

C. CONCLUSIONS

Both the thinning algorithm and the Poisson decomposition and gap-statistics algorithm include two general types of operations: a "generating" process and a "second stage". For the Poisson decomposition and gap-statistics algorithm, the generating process is the non-homogeneous Poisson process with degree-one exponential polynomial intensity function.

Algorithm (Program Name)	Computer Memory in Bytes
1. Poisson Decomposition and Gap-Statistics (DEGTWO)	87,752
2. Basic Thinning Algorithm (NHPTHN)	70,134
3. Modified Thinning Algorithm (NHPP)	114,110
4. Thinning Algorithm Which Generates Next Point (NHPNXT)	30,104

TABLE VI

COMPUTER MEMORY REQUIREMENTS

For the thinning algorithm (as implemented herein) the generating process is homogeneous Poisson.

The second stage for the Poisson decomposition and gap-statistics algorithm is the actual decomposition and generation of variates by the conditioning-acceptance-rejection method. For the thinning algorithm, the second stage consists of the thinning of the points in the bounding process. Thus one algorithm generates events and adds more events from a second process while the other generates events and subtracts some out.

The strongest point in the Poisson decomposition and gap-statistics algorithm is the highly efficient generation of the events in the degree-one exponential polynomial intensity function process. This is done with the gap-statistics algorithm which is two to five times faster than the thinning algorithm for this type of process (see Appendix A). At the same time, the conditioning-acceptance-rejection routine is relatively quite inefficient.

There are many considerations in predicting the relative success of the two algorithms, i.e. which will be faster in a given situation. For example, the Poisson decomposition and gap-statistics algorithm is affected by factors such as whether or not partitioning is required, the percentage of the total number of variates which come from the degree-one exponential polynomial process, and whether time reversal is required. For the thinning algorithm, the fraction

of the lower bound divided by the upper bound $\underline{\lambda}/\lambda^*$, would seem to be a good indicator of success.

For the test cases considered, however, the only consistent indicator was the expected number of events in the non-homogeneous Poisson process being simulated. The smaller the number of events in the non-homogeneous Poisson process being simulated, the better the relative performance of the thinning algorithm over the Poisson decomposition and gap-statistics algorithm. Thus it appears that each algorithm has a fixed and variable part in terms of time. The thinning algorithm has a shorter "setup" cost in terms of time but the variable cost or "cost per additional variate" seems to be smaller for the Poisson decomposition and gap-statistics algorithm.

The exact cause of this phenomenon is not known although it appears to be centered in the conditioning-acceptance-rejection routine.

In the larger spectrum of non-homogeneous Poisson process generation, it seems clear that the thinning algorithm is the best all-around method available.

D. RECOMMENDATIONS

Two specific areas for further study are recommended.

First, the thinning algorithm as implemented here uses only homogeneous Poisson processes for bounding. It might be worthwhile to investigate the possibility of using other processes, such as non-homogeneous Poisson processes with degree-one exponential polynomial intensity functions, as

bounding processes. This would allow the efficient gap-statistics algorithm to be utilized although function evaluation would in general become more time consuming.

The second area is in finding the optimum method for generating the degree-two exponential polynomial class of intensity function. These will undoubtedly remain of interest due to their statistical properties. Here, it seems clear that the best features of the two algorithms can be combined. Specifically, the Poisson decomposition and gap-statistics algorithm can be modified to use thinning rather than conditioning-acceptance-rejection for generating the points in the difference function process, i.e. the process with intensity function $\lambda_D = \lambda(t) - \lambda_L(t)$. Also the criterion for the decomposition might preferably be that the intensity function of the remainder be monotonically increasing. This would make it easy to find the upper bound for the function, and the most efficient version of the thinning algorithm, that where the number of computations of the intensity function is minimized, could be used.

APPENDIX A

GENERATION OF DEGREE-ONE EXPONENTIAL POLYNOMIAL INTENSITY FUNCTIONS

The generating process for the Poisson decomposition and gap-statistics algorithm is a non-homogeneous Poisson process with degree-one exponential polynomial intensity function. This is generated by using the gap-statistics method, which is subroutine NHPP2 in the DEGTWO program (see listing below).

To determine the relative speed of the thinning algorithm compared to the gap-statistics algorithm, two simple degree-one exponential polynomial intensity functions were developed and simulated.

Table VII presents the results. Case A is a monotone decreasing intensity function, $\lambda(t) = \exp(3.4 - 0.02 \cdot t)$ over the interval (0,100]. Case B is a monotone increasing intensity function, $\lambda(t) = \exp(.693 + 0.03 \cdot t)$ over the interval (0,50].

Results show that the gap-statistics algorithm is from two and a half (Case B) to four and a half (Case A) times faster than the thinning algorithm.

CASE	E(N)	PROGRAM	$\overline{N}$	TIME (SEC)	RATIO (NHPP ÷ NHPP2)
A	1295	NHPP	1296	26.9560	4.488
		NHPP2	1294	6.0058	
B	232	NHPP	232	13.2448	2.434
		NHPP2	232	5.4415	

Program NHPP is the final thinning program

Program NHPP2 is the gap-statistics algorithm program

TABLE VII

COMPARISON OF GAP-STATISTICS AND THINNING ALGORITHMS FOR GENERATION OF DEGREE-ONE EXPONENTIAL POLYNOMIAL INTENSITY FUNCTION PROCESSES

APPENDIX B

RESULTS OF EFFICIENCY MODIFICATIONS

This appendix presents, in tabular form, the results of comparison of the programming modifications listed in Section V.

Table VIII shows the effects of utilizing lower bounds for the intensity function. Test conditions include:

1. Use of uniform thinning variates
2. Recycling of thinning variates
3. Use of arrays of variates

Table IX shows the gains realized by employing exponential thinning variates in contrast to uniform thinning variates for exponential polynomial intensity functions. Test conditions include:

1. Use of arrays of variates
2. Recycling of thinning variates
3. Use of lower bounds for intensity function

Table X shows the results of recycling versus no recycling where uniform variates are used for thinning while Table XI shows the same comparison when exponential variates are used for thinning. For both cases, test conditions include:

1. Use of arrays of random variates
2. Use of lower bounds for intensity function

Case	E(N)	Algorithm	Replications Per Package	M*	$\overline{N}$	Time(sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	2	1598	26.5771	.7751
		B	30	2	1604	34.2876	
III	526	A	75	2	526	17.4560	.7374
		B	75	2	527	23.6733	
IV	174	A	100	2	175	10.7023	.9052
		B	100	2	175	11.8233	
V	208	A	100	2	206	12.9191	.8428
		B	100	2	207	15.3282	
VI	124	A	100	2	124	8.0800	.9528
		B	100	2	124	8.4800	

Note: Case I not run due to inefficiency of thinning algorithm (see Section IV.C).

* M = number of replications of each package

Algorithm A: Lower Bound Utilized
Algorithm B: Lower Bound = 0.0

TABLE VIII

LOWER BOUND VERSUS NO LOWER BOUND; UNIFORM THINNING VARIATES

Case	E(N)	Algorithm	Replications Per Package	M*	$\bar{N}$	Time(sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	30	1614	25.7170	.9034
		B	30	30	1578	28.4658	
III	526	A	75	30	527	18.5615	.9175
		B	75	30	526	20.2297	
IV	174	A	100	30	175	10.5615	.9049
		B	100	30	175	11.6719	
V	208	A	100	30	208	11.3416	.9105
		B	100	30	208	12.4561	
VI	124	A	100	30	124	7.4170	.9016
		B	100	30	124	8.2265	

Note: Case I not run due to inefficiency of thinning algorithm (see Section IV.C).

* M = number of replications of each package

Algorithm A: Exponential Thining Variates (Program NHPP, NTYPE = 1)
Algorithm B: Uniform Thinning Variates (Program NHPP, NTYPE = 0)

TABLE IX

EXPONENTIAL THINNING VARIATES VERSUS UNIFORM THINNING VARIATES FOR EXPONENTIAL POLYNOMIAL INTENTISY FUNCTION

Case	E(N)	Algorithm	Replications Per Package	M*	$\bar{N}$	Time(sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	4	1572	28.6048	.9835
		B	30	4	1613	29.0842	
III	526	A	75	6	525	20.1999	.9697
		B	75	6	525	20.8305	
IV	174	A	100	11	174	11.7644	.9894
		B	100	11	178	11.8902	
V	208	A	100	12	207	11.2612	.9772
		B	100	12	208	11.5240	
VI	124	A	100	18	124	7.5126	.9811
		B	100	18	124	7.6574	

Note: Case I not run due to inefficiency of thinning algorithm (see Section IV.C).

* M = number of replications of each package

Algorithm A: Recycling Used (NHPP with NTYPE = 0)
Algorithm B: No Recycling (Modified NHPP with NTYPE = 0)

TABLE X

RECYCLING VERSUS NO RECYCLING; UNIFORM THINNING VARIATES

Case	E(N)	Algorithm	Replications Per Package	M*	$\overline{N}$	Time(sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	30	1614	25.2592	.9498
		B	30	30	1604	26.5950	
III	526	A	75	30	527	20.5533	.9349
		B	75	30	524	21.9927	
IV	174	A	100	30	175	11.0176	1.0176
		B	100	30	175	10.8271	
V	208	A	100	30	207	10.0599	.9913 (.9952)*
		B	100	30	208	10.1479	
VI	124	A	100	30	125	7.5407	1.0531 (1.0434)*
		B	100	30	124	7.1601	

Note: Case I not run due to inefficiency of thinning algorithm (see Section IV.C).

* M = number of replications of each package

Algorithm A: Recycling Used (NHPP with NTYPE = 1)
Algorithm B: No Recycling (Modified NHPP with NTYPE = 1)

* 2σ vice 4σ excess exponentials called for recycling

TABLE XI

RECYCLING VERSUS NO RECYCLING; EXPONENTIAL THINNING VARIATES

Table XII presents the results of incorporating all of the programming improvements into program NHPP. The final thinning program, NHPP, is compared to the basic thinning program without modifications, NHPTHN. The essential differences are:

NHPP (final program)	NHPTHN (basic program)
Arrays of variates generated	Variates generated one at a time
Exponential variates used for thinning	Uniform variates used for thinning
Lower bound of intensity functions utilized	Lower bound = 0.0
Thinning variates recycled	No recycling used

Case	E(N)	Algorithm	Replications Per Package	M*	$\bar{N}$	Time(sec) Per Package	Ratio of Times (A ÷ B)
II	1612	A	30	30	1612	25.5728	.605
		B	30	30	1612	42.2534	
III	526	A	75	30	526	18.2078	.576
		B	75	30	526	31.6105	
IV	174	A	100	30	175	10.4186	.720
		B	100	30	175	14.4654	
V	208	A	100	30	208	11.1811	.676
		B	100	30	207	16.5307	
VI	124	A	100	30	124	7.3525	.765
		B	100	30	124	9.6104	

Note: Case I not run due to inefficiency of thinning algorithm (see Section IV.C).

* M = number of replications of each package

Algorithm A: Final Thinning Program (NHPP)
Algorithm B: Basic Thinning Program (NHPTHN)

TABLE XII

FINAL THINNING PROGRAM VERSUS BASIC THINNING PROGRAM

```
C   SUBROUTINE NHPP
C
C      SUBROUTINE NHPP
C
C      PURPOSE
C         SIMULATES A NON-HOMOGENEOUS POISSON PROCESS
C         WITH INTENSITY FUNCTION FCN(X) OVER A GIVEN
C         INTERVAL USING THE THINNING ALGORITHM.
C
C      USAGE
C         CALL NHPP(IS,EL,ER,UB,XMIN,NTYPE,N,IER)
C
C
C      DESCRIPTION OF PARAMETERS
C         IS - RANDOM NUMBER SEED.  ANY INTEGER WITH
C              NINE OR LESS DIGITS.
C         EL - LEFT END POINT OF INTERVAL
C         ER - RIGHT END POINT OF INTERVAL
C         UB - UPPER BOUND OF THE INTENSITY FUNCTION
C              FCN(X) OVER THE INTERVAL (EL,ER).  THE
C              CLOSER UB IS TO THE LUB, THE MORE
C              EFFICIENT THE PROGRAM
C         XMIN - LOWER BOUND OF THE INTENSITY FUNCTION
C              OVER THE INTERVAL.  THE CLOSER XMIN IS TO
C              THE GLB, THE MORE EFFICIENT THE PROGRAM.
C         NTYPE - 1 IF THE INTENSITY FUNCTION IS
C                   EXPONENTIAL POLYNOMIAL, I.E. OF
C                   THE FORM EXP(A + A *X + A2*X**2 +...)
C               0 OTHERWISE
C         N - THE TOTAL NUMBER OF EVENTS IN THE
C             NON-HOMOGENEOUS POISSON PROCESS
C         IER - ERROR FLAG.  IER HAS FOLLOWING MEANINGS:
C               1...ER IS LESS THAN EL
C               2...UB IS NON-POSITIVE
C               3...XMIN IS NEGATIVE
C               4...MORE THAN 5000 EVENTS REQUIRED FOR
C                   BOUNDING PROCESS: STORAGE CAPACITY
C                   EXCEEDED.
C               5...XMIN IS GREATER THAN UB
C
C      COMMENTS
C         CALLING PROGRAM MUST HAVE A COMMON REGION,
C         DONNA, OF DIMENSION(5000)
C
C         EXAMPLE:  DIMENSION T(5000)
C                   COMMON/DONNA/T
C
C         TIMES TO EVENTS WILL BE STORED IN
C         CELLS T(1) THROUGH T(N)
C
C         THE INTENSITY FUNCTION IS USER SUPPLIED.
C         IF THE INTENSITY FUNCTION IS NOT EXPONENTIAL
C         POLYNOMIAL, I.E. IF NTYPE = 0, THE ENTIRE
C         INTENSITY FUNCTION IS EVALUATED.
C
C         EXAMPLE:  FUNCTION FCN(X)
C                   A = 1.0
C                   FCN = B*X
C                   RETURN
C                   END
C
C         IF THE INTENSITY FUNCTION IS EXPONENTIAL
C         POLYNOMIAL, I.E. NTYPE = 1, ONLY THE
C         POLYNOMIAL IS EVALUATED.
C
C         EXAMPLE:  FUNCTION FCN(X)
C                   A = 1.0
C                   A1 = 0.5
C                   A2 = 0.05
C                   FCN = A + A1*X + A2*X**2
C                   RETURN
```

```
C                   END
C
C        PROGRAMMER:  LCDR JOHN SCOTT REDD, USN
C        SEPTEMBER 1978
          SUBROUTINE NHPP (IS,EL,ER,UB,XMIN,NTYPE,N,IER)
          DIMENSION U(5000), TIMES(5000), TT(5000), EE(5000)
          COMMON /DONNA/ TT
          COMMON /ANNE/ TIMES,EE
          EXTERNAL FCN
          CALL OVFLOW
C
C      INITIALIZE VARIABLES
C
          IER = 0
          ZK = .0001
          ZL = 1OE-6
          PCTMIN = .05
C
C      CHECK FOR PARAMETER ERRORS
C
          IF (EL.GE.ER) IER = 1
          IF (UB.LE.0.0) IER = 2
          IF (XMIN.LT.0.0) IER=3
          IF (UB-XMIN.LT.ZL) IER=5
          IF (IER.NE.0) GO TO 14
C
C      GENERATE POINTS IN HOMOGENEOUS POISSON PROCESS
C      WITH RATE = UB
C
          CALL HPP (IS,EL,ER,UB,NTYPE,NSTAR,NLEFT,NEXP,IER)
          IF (IER.EQ.4) GO TO 14
C
C      IS INTENSITY FUNCTION OTHER THAN LOG QUADRATIC?
C
          IF (NTYPE.EQ.0) GO TO 9
C
C      LOG QUADRATIC INTENSITY FUNCTION
C
C      DOES IT HAVE A MINIMUM OR IS MINIMUM LESS THAN
C      PCTMN OF MAX?
C
          PCT = XMIN/UB
          IF (PCT.LT.PCTMIN) GO TO 6
C
C      USE MINIMUM
C
C
C
C      COMPUTE EXPECTED NUMBER OF EXPONENTIALS NEEDED, TAKING
C      INTO ACCOUNT REUSEING OF THOSE IN MIN
C
          P = 1.0-PCT
          Q = PCT
          XNSTAR = FLOAT(NSTAR)
          NCHK = IFIX(XNSTAR*P+4.0*SQRT(XNSTAR*P*Q))
          NCHK = MINO(NCHK,NSTAR)
          CALL REORD (NCHK,NEXP,NLEFT)
          NCALL = NCHK-NLEFT
          IF (NCALL.LE.0) GO TO 1
          CALL SEXPON (IS,EE,NCALL)
        1 K = 0
          KK = 1
          EB = EE(KK)
          UBLN = ALOG(UB)
          BND = ALOG(UB/XMIN)
          KKK = 0
C
C      I COUNTS THE HPP EVENTS
C      K COUNTS THE NHPP EVENTS
C      KK COUNTS THE CURRENT UNIT EXPONENTIALS (FOR THINNING)
C      KKK COUNTS THE TOTAL NUMBER OF UNIT EXPONENTIALS
C      (FOR THINNING)
C
```

```
      DO 5 I=1,NSTAR
      IF (EB.LT.BND) GO TO 2
      K = K+1
      TT(K) = TIMES(I)
      EB = EB-BND
      GO TO 5
    2 VAL = -FCN(TIMES(I))+UBLN
      IF (EB.LT.VAL) GO TO 3
      K = K+1
      TT(K) = TIMES(I)
    3 KK = KK+1
C
C  CHECK TO SEE IF MORE UNIT EXPON NEEDED
C
      IF (KK.LE.NCHK) GO TO 4
C
C  GENERATE MORE UNIT EXPONENTIALS FOR THINNING
C
      KKK = KKK+KK
      PN = P*FLOAT(NSTAR-KKK)
      NEB = MAX0(1,IFIX(PN+4.0*SQRT(PN*Q)))
      CALL SEXPON (IS,EE,NEB)
      KK = 1
      NCHK = NEB
    4 EB = EE(KK)
    5 CONTINUE
      N = K
      GO TO 14
C
C  LOG QUADRATIC WITH NO MINIMUM
C
    6 CONTINUE
      CALL REORD (NSTAR,NEXP,NLEFT)
      NCALL = NSTAR-NLEFT
      IF (NCALL.LE.0) GO TO 7
      CALL SEXPON (IS,EE,NCALL)
C
C  SET VARIABLES
C
    7 K = 0
      UBLN = ALOG(UB)
C
C  I COUNTS HPP EVENTS
C  K COUNTS NHPP EVENTS
C
      DO 8 I=1,NSTAR
      VAL = -FCN(TIMES(I))+UBLN
      IF (EE(I).LT.VAL) GO TO 8
      K = K+1
      TT(K) = TIMES(I)
    8 CONTINUE
      N = K
      GO TO 14
C
C  INTENSITY FUNCTION IS NOT LOG QUADRATIC
C
C  DOES IT HAVE A MINIMUM OR IS MINIMUM LESS THAN
C  PCTMN OF MAX?
C
    9 PCT = XMIN/UB
      IF (PCT.LT.PCTMIN) GO TO 12
C
C  USE MINIMUM
C
C  INITIALIZE VARIABLES
C
      NUNIF = NSTAR
      CALL SRAND (IS,U,NUNIF)
      K = 0
      FF = 1.0/UB
C
C  I COUNTS HPP EVENTS
```

```
C   K COUNTS NHPP EVENTS
C
       DO 11 I=1,NSTAR
       IF (U(I).GT.PCT) GO TO 10
       K = K+1
       TT(K) = TIMES(I)
       GO TO 11
    10 VAL = FCN(TIMES(I))*FF
       IF (U(I).GT.VAL) GO TO 11
       K = K+1
       TT(K) = TIMES(I)
       GO TO 11
    11 CONTINUE
       N = K
       GO TO 14
C
C   NO MINIMUM
C
    12 NUNIF = NSTAR
       CALL SRAND (IS,U,NUNIF)
       K = 0
       FF = 1.0/UB
       DO 13 I=1,NSTAR
       VAL = FCN(TIMES(I))*FF
       IF (U(I).GT.VAL) GO TO 13
C
C   ACCEPT POINT
C
       K = K+1
       TT(K) = TIMES(I)
C
C
C   REJECT POINT
C
    13 CONTINUE
       N = K
    14 RETURN
       END
C   SUBROUTINE HPP
C
C   SUBROUTINE HPP GENERATES POINTS IN A HOMOGENEOUS
C   POISSON PROCESS WITH INTENSITY FUNCTION = UB
C
       SUBROUTINE HPP(IS,EL,ER,UB,NTYPE,NSTAR,NLEFT,NEXP,IER)
       DIMENSION TIMES(5000), EE(5000)
       COMMON /ANNE/ TIMES,EE
C
C   INITIALIZE VARIABLES
C
       EXMEAN = 1.0/UB
       NEXP = IFIX(UB*(ER-EL)+4.0*SQRT(UB*(ER-EL)))
       IF (NEXP.GT.5000) NEXP=5000
       CALL SEXPON (IS,EE,NEXP)
       SUM = EL
       ISTART = 0
       ISTOP = NEXP
C
       DO 1 JJ=1,NEXP
       E = EXMEAN*EE(JJ)
       SUM = SUM+E
       TIMES(JJ) = SUM
       IF (SUM.LT.ER) GO TO 1
       NSTAR = JJ-1
       GO TO 5
     1 CONTINUE
C
C   EXCEPTIONAL NUMBER OF POINTS NEEDED
C
     2 ISTART = ISTART+JJ
       NEXP = IFIX(UB*(ER-SUM)+4.0*SQRT(UB*(ER-SUM)))
       IF (NEXP.EQ.0) NEXP=1
```

```
      ISTOP = ISTART+NEXP
      IF (ISTOP.GT.5000) GO TO 4
      CALL SEXPON (IS,EE,NEXP)
      DO 3 JJ=1,NEXP
      E = EXMEAN*EE(JJ)
      SUM = SUM+E
      KK = JJ+ISTART
      TIMES(KK) = SUM
      IF (SUM.LT.ER) GO TO 3
      NSTAR = KK-1
      GO TO 5
    3 CONTINUE
C
      GO TO 2
C
C   MORE THAN 5000 EXPONENTIALS NEEDED
C
    4 IER = 4
      GO TO 6
C
C   CALCULATE NUMBER OF EXPONENTIALS NOT USED
C
C
    5 NLEFT = NEXP-JJ
    6 RETURN
      END
C...SUBROUTINE REORD
C    SUBROUTINE REORD REORDERS EXPONENTIALS LEFT OVER FROM
C    SUBROUTINE HPP FOR USE AS THINNING VARIATES IN NHPP
C
      SUBROUTINE REORD (NCHK,NEXP,NLEFT)
      DIMENSION EE(5000), TIMES(5000)
      COMMON /ANNE/ TIMES,EE
C
C   ARE ENOUGH EXPONENTIALS LEFT OVER FROM HPP?
C
      IF (NLEFT.GE.NCHK) GO TO 4
C
C   MORE EXPONENTIALS NEEDED
C
C   REORDER TOP TO BOTTOM OR VICE VERSA?
C
      IF (NEXP.LT.NCHK) GO TO 2
C
C   REORDER BOTTOM TO TOP
C
      N1 = NCHK-NLEFT
      N2 = NEXP-NLEFT
C
      DO 1 I=1,NLEFT
      J = N1+I
      K = N2+I
      EE(J) = EE(K)
    1 CONTINUE
      GO TO 6
C
C   REORDER TOP TO BOTTOM
C
    2 NCHKO = NCHK+1
      NEXPO = NEXP+1
      DO 3 I=1,NLEFT
      J = NCHKO-I
      K = NEXPO-I
      EE(J) = EE(K)
    3 CONTINUE
      GO TO 6
C
C   ENOUGH LEFT OVER FROM HPP FOR ALL THINNING
C
    4 N1 = NEXP-NLEFT
      DO 5 I=1,NCHK
```

```
      J = N1+I
      EE(I) = EE(J)
    5 CONTINUE
    6 RETURN
      END
```

```
C
C
C     SUBROUTINE NHPTHN
C
C     PURPOSE
C        SIMULATES A NONHOMOGENEOUS POISSON PROCESS
C        WITH INTENSITY FUNCTION FCN(X) USING .
C        THE THINNING ALGORITHM
C
C     USAGE
C        CALL NHPTHN(IS,EL,ER,UB,N,IER)
C
C     DESCRIPTION OF PARAMETERS
C        IS - RANDOM NUMBER SEED.  ANY INTEGER WITH NINE
C             OR LESS DIGITS.
C        EL - LEFT END POINT OF INTERVAL
C        ER - RIGHT END POINT OF THE INTERVAL
C        UB - UPPER BOUND OF THE INTENSITY FUNCTION,FCN(X),
C             OVER THE INTERVAL (EL,ER).  THE CLOSER UB IS
C             TO THE LEAST UPPER BOUND,LUB, THE MORE
C             EFFICIENT THE PROGRAM.  UB MUST BE STRICTLY
C             POSITIVE
C        N  - THE TOTAL NUMBER OF EVENTS IN THE
C             NON-HOMOGENEOUS POISSON PROCESS.
C        IER - ERROR FLAG.  IER HAS FOLLOWING MEANINGS:
C           1...ER IS LESS THAN EL
C           2...UB IS NON-POSITIVE
C
C     COMMENTS
C        CALLING PROGRAM MUST HAVE A COMMON REGION,
C        SCOTT, OF DIMENSION(5000)
C
C           EXAMPLE:  DIMENSION T(5000)
C                     COMMON/SCOTT/T
C
C        TIMES TO EVENTS WILL BE STORED IN CELLS
C        T(I) THROUGH T(M).
C
C      PROGRAMMER:  LCDR JOHN SCOTT REDD, USN
C                   AUGUST 1978
C
         SUBROUTINE NHPTHN (IS,EL,ER,UB,N,IER)
         DIMENSION TIMES(5000), TTT(5000)
         COMMON /SCOTT/ TTT
         EXTERNAL FCN
         CALL OVFLOW
C        INITIALIZE VARIABLES
         IER = 0
         IF (EL.GE.ER) IER = 1
         IF (UB.LE.0.0) IER = 2
         IF (IER.NE.0) RETURN
C
C     GENERATE POINTS IN HOMOGENEOUS POISSON PROCESS WITH RATE
C     = UB.
C
         EXMEAN = 1.0/UB
         I = 1
         SUM = EL
       1 CONTINUE
         CALL EXPON (IS,E,1)
         E = E*EXMEAN
         SUM = SUM+E
         IF (SUM.GT.ER) GO TO 2
         TIMES(I) = SUM
         I = I+1
         GO TO 1
       2 CONTINUE
         NSTAR = I-1
C
C     COMMENCE THINNING THE NSTAR POINTS
C
         F1 = 1.0/UB
```

```
      K = 0
      IF (NSTAR.EQ.0) GO TO 4
C
      DO 3 I=1,NSTAR
      CALL SRAND (IS,U,1)
      RATIO = FCN(TIMES(I))*F1
      IF (U.GT.RATIO) GO TO 3
      K = K+1
      TTT(K) = TIMES(I)
    3 CONTINUE
C
      N = K
      RETURN
    4 N = 0
      RETURN
      END
```

```
C   SUBROUTINE NHPOAT
C
C
C  SUBROUTINE NHPOAT
C
C    PURPOSE
C      SIMULATES A NON-HOMOGENEOUS POISSON PROCESS
C      WITH INTENSITY FUNCTION FCN(X) OVER THE
C      INTERVAL (EL,ER) USING THE ONE-AT-A-TIME
C      THINNING ALGORITHM
C
C...USAGE
C     CALL NHPOAT(IS,EL,ER,UB,N,IER)
C
C...DESCRIPTION OF PARAMETERS
C
C     IS - RANDOM NUMBER SEED.  ANY INTEGER WITH NINE
C          OR LESS DIGITS.
C     EL - LEFT END POINT OF INTERVAL
C     ER - RIGHT END POINT OF THE INTERVAL
C     UB - UPPER BOUND OF THE INTENSITY FUNCTION,FCN(X),
C          OVER THE INTERVAL (EL,ER).  THE CLOSER UB IS
C          TO THE LEAST UPPER BOUND,LUB, THE MORE
C          EFFICIENT THE PROGRAM.  UB MUST BE STRICTLY
C          POSITIVE
C     N  - THE TOTAL NUMBER OF EVENTS IN THE
C          NON-HOMOGENEOUS POISSON PROCESS.
C     IER - ERROR FLAG.   IER HAS FOLLOWING MEANINGS;
C         1...ER IS LESS THAN EL
C         2...UB IS NON-POSITIVE
C...COMMENTS
C     CALLING PROGRAM MUST HAVE A COMMON REGION, DONNA, OF
C       DIMENSION(5000)
C
C          EXAMPLE:  DIMENSION TT(5000)
C                    COMMON/DONNA/TT
C
C      TIMES TO EVENTS WILL BE STORED IN CELLS
C      T(1) THROUGH T(N)
C
      SUBROUTINE NHPOAT (IS,EL,ER,UB,N,IER)
      DIMENSION TT(5000)
      COMMON /DONNA/ TT
      EXTERNAL FCN
      CALL OVFLOW
C...INITIALIZE VARIABLES
      IER = 0
      IF (EL.GE.ER) IER = 1
      IF (UB.LE.0.0) IER = 2
      IF (IER.NE.0) RETURN
C
      I = 1
      EXMEAN = 1.0/UB
      SUM = EL
C
C   GENERATE POINT IN BOUNDING PROCESS
C
    1 CONTINUE
      CALL EXPON (IS,E,1)
      E = E*EXMEAN
      SUM = SUM+E
      IF (SUM.GT.ER) GO TO 2
C
C   THIN THE POINT
C
      CALL RANDOM (IS,U,1)
      RATIO =  FCN(SUM)*EXMEAN
      IF (U.GT.RATIO) GO TO 1
      TT(I) = SUM
      I = I+1
      GO TO 1
    2 N = I-1
```

```
RETURN
END
```

```
C     SUBROUTINE NHPNXT
C
C        SUBROUTINE NHPNXT GENERATES THE TIME OF THE NEXT
C        EVENT IN A NON-HOMOGENEOUS POISSON PROCESS WITH
C        RATE FUNCTION FCN(X) (USER SUPPLIED).
C
C        USAGE:
C           CALL NHPNXT(IS,UB,GLB,XLAST,ER,XNEXT,IER)
C
C        DESCRIPTION OF PARAMETERS:
C           IS     - RANDOM NUMBER SEED.  ANY INTEGER WITH
C                    NINE OR LESS DIGITS.
C           UB     - UPPER BOUND OF THE INTENSITY FUNCTION
C                    OVER THE INTERVAL (XLAST,ER).
C           GLB    - GREATEST LOWER BOUND OF THE INTENSITY
C                    FUNCTION OVER THE INTERVAL (XLAST,ER).
C                    SET = 0 IF UNKNOWN.
C           XLAST  - THE TIME OF THE LAST EVENT IN THE PROCESS
C           ER     - RIGHT END POINT OF THE INTERVAL
C           XNEXT  - THE NEXT POINT IN THE PROCESS.  IF THERE ARE
C                    NO MORE POINTS IN THE INTERVAL (XLAST,ER),
C                    XNEXT IS ASSIGNED THE VALUE ER + 1.0 AND
C                    IER IS SET AT 5.
C           IER    - ERROR FLAG.  IER HAS THE FOLLOWING MEANINGS:
C                    1...UB IS NON-POSITIVE
C                    2...XLAST IS GREATER THAN ER
C                    3...GLB IS NEGATIVE
C                    5...XNEXT IS GREATER THAN ER
C
C        COMMENTS
C           THE INTENSITY FUNCTION, FCN, IS USER SUPPLIED.
C           EXAMPLE:   FUNCTION FCN(X)
C                      FCN = 1.0 + EXP(-X)
C                      RETURN
C                      END
C
C
C        PROGRAMMER:   LCDR JOHN SCOTT REDD, USN
C                      AUG 1978
C
C
C
         SUBROUTINE NHPNXT (IS,UB,GLB,XLAST,ER,XNEXT,IER)
         EXTERNAL FCN
         CALL OVFLOW
         DATA EXPO/0.0/,U/0.0/
         RMIN = GLB/UB
         IER = 0
         IF (UB.LE.0.0) IER = 1
         IF (XLAST.GE.ER) IER=2
         IF (GLB.LT.0.0) IER=3
         IF (IER.EQ.1.OR.IER.EQ.2) RETURN
         IF (IER.EQ.3) GLB = 0.0
C
C     GENERATE E*(I,J);  CHECK TO SEE IF ADDITION OF E*(I,J)
C        EXCEEDS ER
C
         EXMEAN = 1.0/UB
         XNEW = XLAST
C
C     GENERATE ONE EXPONENTIAL AND SCALE
C
      1  CALL EXPON (IS,EXPO,1)
         EXPO = EXPO*EXMEAN
         XNEW = XNEW+EXPO
         IF (XNEW.GT.ER) GO TO 3
C
C     GENERATE UNIFORM(0,1) THINNING VARIATE
C
         CALL RANDOM (IS,U,1)
C
C     TEST LOWER BOUND FOR THINNING
```

```
C
      IF (U.LE.RMIN) GO TO 2
C
C     TEST FOR THINNING
C
      RATIO = FCN(XNEW)/UB
      IF (U.LE.RATIO) GO TO 2
      GO TO 1
C
C     ARRIVAL HERE INDICATES SUCCESSFUL THINNING
C
    2 XNEXT = XNEW
      RETURN
C
C     ARRIVAL HERE INDICATES NO MORE POINTS IN INTERVAL
C
    3 IER = 5
      XNEXT = ER+1.0
      RETURN
      END
```

```
C    SUBROUTINE DEGTWO
C
C      SUBROUTINE DEGTWO
C
C      PURPOSE
C         SIMULATES A NON-HOMOGENEOUS POISSON PROCESS WITH
C        QUADRATIC EXPONENTIAL INTENSITY FUNCTION OVER
C        A GIVEN INTERVAL USING THE POISSON-DECOMPOSITION
C        AND GAP STATISTIC ALGORITHM.
C
C      USAGE
C        CALL DEGTWO(IS,A,A1,A2,EL,ER,II,N,IER)
C
C      DESCRIPTION OF PARAMETERS
C         IS - RANDOM NUMBER SEED.  ANY INTEGER WITH NINE
C              OR LESS DIGITS.
C         A - CONSTANT IN INTENSITY FUNCTION
C         A1  - 1ST DEGREE COEFF IN INTENSITY FUNCTION.
C         A2  - 2ND DEGREE COEFF IN INTENSITY FUNCTION.
C         EL  - LEFT END POINT OF INTERVAL.
C         ER  - RIGHT END POINT OF INTERVAL
C         II  - 0 FOR TIMES OF EVENTS.
C               1 FOR TIMES BETWEEN EVENTS.
C         N   - A VECTOR OF LENGTH 5.   N(1) THROUGH N(4)
C             CONTAIN NUMBERS OF EVENTS FROM VARIOUS
C             COMPONENTS OF THE DECOMPOSED INTENSITY
C             FUNCTION.  N(5) CONTAINS THE TOTAL NUMBER
C             OF EVENTS IN THE NON-HOMOGENEOUS POISSON
C            PROCESS.
C
C      COMMENTS
C         CALLING PROGRAM MUST HAVE A COMMON REGION, HOLD,
C         OF DIMENSION (5000),AND AN INTEGER ARRAY OF
C         DIMENSION (5).
C
C         EXAMPLE:  DIMENSION T(5000),N(5)
C                   COMMON/HOLD/T
C
C         CALLING PROGRAM MUST CONTAIN THE FOLLWOING
C         ASSIGNMENT STATEMENT:
C
C            M=N(5)
C
C         CALLING PROGRAM MUST USE THE FOLLOWING JCL CARDS
C
C           //  EXEC  FORTCLG,IMSL = DP
C           //FORT.SYSIN  DD  *
C
C         TIMES TO EVENTS OR TIMES BETWEEN EVENTS WILL BE
C         STORED IN CELLS T(1) THROUGH T(M).
C
C
      SUBROUTINE DEGTWO (IS,A,A1,A2,EL,ER,II,N,IER)
      DIMENSION TIMES(5000), T(5000), N(5), P(5)
      COMMON /MIKE/ TIMES/HOLD/T
      CALL OVFLOW
C
C    INITIALIZE VARIABLES
C
      P(1) = A
      P(2) = A1
      P(3) = A2
      P(4) = 0.
      P(5) = 0.
C
C
      DO 1 I=1,5
    1 N(I) = 0
C
C
C
C    IF RATE FUNCTION IS LESS THAN DEGREE TWO,
C    USE NHPP2 ROUTINE ONLY
```

```
C
      IF (A2.EQ.0.) GO TO 2
      GO TO 4
    2 CALL NHPP2 (IS,EL,ER,A,A1,II,N1,IER)
      N(5) = N1
      IF (N1.EQ.0) RETURN
C
C
      DO 3 I=1,N1
      TIMES(I) = T(I)
    3 CONTINUE
C
      RETURN
C
C   DETERMINE COEFFICIENTS FOR MODIFIED
C   DEGREE ONE RATE FUNCTION
C
    4 TEST = -A1/(2.*A2)
      TINT = ER-EL
      IF (A1.GE.0..AND.A2.GT.0.) GO TO 5
      GO TO 6
    5 B = A-A2*TINT**2
      B1 = A1+2.*A2*TINT
      GO TO 10
    6 B = A
      IF ((A1.LE.0..AND.A2.LT.0.).OR.(A1.GT.0..AND.A2.LT.0..
     1TINT)) GO TO 7
      GO TO 8
    7 B1 = A1+A2*TINT
      GO TO 10
    8 IF (A1.GT.0..AND.A2.LT.0..AND.TEST.LT.TINT) GO TO 9
      B1 = A1
      GO TO 10
    9 B1 = A1/2.
C
C   MUST THE INTERVAL BE PARTITIONED?
C
   10 IF (A1*A2.LT.0..AND.TEST.LT.TINT) GO TO 11
      ERNEW = ER
      GO TO 12
   11 ERNEW = TEST+EL
C
C   GENERATE DEGREE ONE NHPP ON INTERVAL
C
   12 BB = B
      BB1 = B1
      CALL NHPP2 (IS,EL,ERNEW,BB,BB1,0,N1,IER)
      N(1) = N1
      IF (N(1).EQ.0) GO TO 14
C
C
      DO 13 I=1,N1
      TIMES(I) = T(I)
   13 CONTINUE
C
C
C   COMPUTE LENGTH OF INTERVAL AND DETERMINE VALUE
C   OF CSTAR FOR USE IN REJECTION ROUTINE
C
   14 Q = ERNEW-EL
      E1 = A
      E2 = A2*Q**2
      E3 = A1*Q
      E4 = A1**2/(4.*A2)
      E5 = A1**2/(2.*A2)
      IF (A1.GE.0..AND.A2.GT.0.) GO TO 15
      IF (A1.LT.0..AND.A2.GT.0..AND.TEST.GE.TINT) GO TO 16
      IF (A1.LT.0..AND.A2.GT.0..AND.TEST.LT.TINT) GO TO 17
      IF (A1.LE.0..AND.A2.LT.0.) GO TO 18
      IF (A1.GT.0..AND.A2.LT.0..AND.TEST.GE.TINT) GO TO 19
      CSTAR = EXP(E1-E4)-EXP(E1)
      GO TO 20
```

```
   15 CSTAR = EXP(E1)-EXP(E1-E2)
      GO TO 20
   16 CSTAR = EXP(E1+E2+E3)-EXP(E1+E3)
      GO TO 20
   17 CSTAR =  EXP(E1-E4)-EXP(E1-E5)
      GO TO 20
   18 CSTAR = EXP(E1)-EXP(E1+E3+E2)
      GO TO 20
   19 CSTAR = EXP(E1+E3+E2)-EXP(E1)
C
C   COMPUTE INTEGRAL OF MODIFIED DEGREE TWO RATE FUNCTION
C   OVER INTERVAL
C
   20 CALL HELP (A,A1,A2,EL,ERNEW,PMTR)
      PMTR = PMTR-(EXP(B)*(EXP(B1*ERNEW)-EXP(B1*EL)))/B1
C
C   IDENTIFY AS FIRST SUBINTERVAL
C
      NOTE = 1
C
C   GENERATE REALIZATION ON POISON (PMTR) VARIATE
C
   21 CALL PVAR (IS,PMTR,M)
      IF (NOTE.EQ.1) GO TO 22
      GO TO 25
C
C   REJECTION ROUTINE USED ON FIRST SUBINTERVAL
C
   22 N(2) = M
      P(4) = B
      P(5) = B1
      IF (N(2).EQ.0) GO TO 24
      CALL REJECT (IS,EL,CSTAR,P,Q,N(2))
C
C
      DO 23 I=1,M
      TIMES(N(1)+I) = T(I)
   23 CONTINUE
C
C
C   HAS THE INTERVAL BEEN PARTITIONED?
C
   24 IF (ERNEW.EQ.ER) GO TO 34
      GO TO 27
C
C   USE REJECTION ROUTINE ON SECOND PART OF INTERVAL
C
   25 N(4) = M
      P(4) = B
      P(5) = B1
C
C   IF NO EVENTS OCCURRED BYPASS REJECTION ROUTINE
C
      IF (N(4).EQ.0) GO TO 35
      Q = ER-ELNEW
      CALL REJECT (IS,ELNEW,CSTAR,P,Q,N(4))
C
C   COPY TIMES OF EVENTS INTO 'TIMES' ARRAY
C
      N4 = N(1)+N(2)+N(3)
C
C
      DO 26 I=1,M
      TIMES(N4+I) = T(I)
   26 CONTINUE
C
C
C   GENERATION OF VARIATES COMPLETE.
C   GO TO ORDERING ROUTINE
C
      GO TO 35
C
```

```
C     INTERVAL PARTITION WAS REQUIRED.  MUST NOW
C     CONSIDER SECOND SUBINTERVAL
C
C     DETERMINE COEFFICIENTS FOR MODIFIED
C     DEGREE ONE RATE FUNCTION
C
   27 IF (A1.GT.0..AND.A2.LT.0.) GO TO 28
      B = A-A2*TINT**2
      B1 = A1+2.*A2*TINT
      GO TO 29
   28 B = A+(A1/2.)*TINT
      B1 = A1/2.+A2*TINT
   29 ELNEW = ERNEW
C
C     GENERATE DEGREE ONE NHPP ON INTERVAL
C
      BB = B
      BB1 = B1
      CALL NHPP2 (IS,ELNEW,ER,BB,BB1,0,N3,IER)
      N(3) = N3
      IF (N(3).EQ.0) GO TO 31
      N3 = N(1)+N(2)
C
C     TRANSFER TIMES BETWEEN ARRAYS
C
C
      DO 30 I=1,N3
      TIMES(N3+I) = T(I)
   30 CONTINUE
C
C
   31 Q = TINT
C
C     DETERMINE VALUE OF CSTAR FOR USE IN
C     THE REJECTION ROUTINE
C
      E2 = A2*Q**2
      E3 = A1*Q
      IF (A1.GT.0..AND.A2.LT.0.) GO TO 32
      CSTAR = EXP(E1-E4)-EXP(E1-E5-E3-E2)
      GO TO 33
   32 CSTAR = EXP(E1-E4)-EXP(E1+E3+E2)
C
C     COMPUTE INTEGRAL OF MODIFIED DEGREE TWO RATE
C     FUNCTION OVER SECOND INTERVAL
C
   33 CALL HELP (A,A1,A2,ELNEW,ER,PMTR)
      PMTR = PMTR-(EXP(B)*(EXP(B1*ER)-EXP(B1*ELNEW)))/B1
C
C    IDENTIFY AS SECOND SUBINTERVAL
C
      NOTE = 2
      GO TO 21
C
C     PARTITION OF INTERVAL NOT REQUIRED.  COMPUTE TOTAL
C     EVENTS AND SUPERPOSE TWO EVENT STREAMS
C
   34 N(5) = N(1)+N(2)
      IF (N(2).EQ.0) GO TO 38
      LBGN = N(1)+1
      JBGN = 1
      CALL COLATE (LBGN,N(5),1)
      GO TO 38
C
C     PARTITION WAS REQUIRED.  DETERMINE
C    AMOUNT OF SORTING NEEDED
C
   35 N(5) = N(1)+N(2)+N(3)+N(4)
      IF (N(2).EQ.0.AND.N(4).EQ.0) GO TO 38
      IF (N(4).EQ.0) GO TO 36
      IF (N(2).EQ.0) GO TO 37
C
```

```
C   MUST SUPERPOSE FOUR EVENT STREAMS
C
      LBGN = N(1)+1
      LFIN = N(1)+N(2)
      CALL COLATE (LBGN,LFIN,1)
      LBGN = LFIN+N(3)+1
      JBGN = LFIN+1
      CALL COLATE (LBGN,N(5),JBGN)
      GO TO 38
C
C   MUST SUPERPOSE FIRST HALF OF ARRAY ONLY
C
   36 N2 = N(1)+N(2)
      LBGN = N(1)+1
      CALL COLATE (LBGN,N2,1)
      GO TO 38
C
C   MUST SUPERPOSE SECOND HALF OF ARRAY ONLY
C
   37 KK = N(1)+N(2)+1
      LBGN = N(1)+N(2)+N(3)+1
      LFIN = N(5)
      CALL COLATE (LBGN,N(5),KK)
      GO TO 38
C
C   ARE TIMES OF EVENTS OR TIMES BETWEEN EVENTS REQUESTED?
C
C
   38 IF (II.EQ.0) RETURN
C
C   CALCULATE TIMES BETWEEN EVENTS
C
      S = TIMES(1)
      TIMES(1) = TIMES(1)-EL
      N5 = N(5)
C
C
      DO 39 I=2,N5
      S1 = TIMES(I)
      TIMES(I) = TIMES(I)-S
      S = S1
   39 CONTINUE
C
      RETURN
      END
C   SUBROUTINE NHPP2 SIMULATES A NON-HOMOGENEOUS
C   POISSON PROCESS WITH A LOG-LINEAR INTENSITY
C   (RATE) FUNCTION
C
      SUBROUTINE NHPP2 (IS,EL,ER,A,A1,II,N,IER)
      DIMENSION T(5000)
      COMMON /HOLD/ T
C
      CALL OVFLOW
C
C   INITIALIZE VARIABLES
C
      IER = 0
      TINT = ER-EL
      A = EXP(A+A1*EL)
C
C  IS THE POISSON PROCESS HOMOGENEOUS?
C
      IF (A1.EQ.0.) GO TO 3
      PAR = (A*(EXP(TINT*A1)-1.))/A1
      IF (A1.GT.0.) GO TO 1
      IFLAG = 3
      GO TO 2
    1 A = A*EXP(TINT*A1)
      A1 = -A1
      IFLAG = 2
C
```

```
C   COMPUTE PARAMETERS OF BOTH POISSON RANDOM VARIABLES
C
    2 THETA = -A/A1
      GO TO 4
C   COMPUTE RATE AND SCALING PARAMETERS FOR HOMOGENEOUS
C   POISSON PROCESS
C
    3 PAR = TINT*A
      IFLAG = 1
      AINVRS = 1./A
C
C   COMPUTE NUMBER OF EXPONENTIAL VARIATES REQUIRED
C
    4 NMAX = PAR+6.*SQRT(PAR)
C
C   IS THIS A HOMOGENEOUS POISSON PROCESS?
C
      IF (IFLAG.EQ.1) GO TO 17
C
C   GENERATE REALIZATION ON POISSON (THETA) VARIATE
C
    5 CONTINUE
      CALL PVAR (IS,THETA,M)
      IF (M.EQ.0) GO TO 7
C
C   CALCULATE TIMES OF EVENTS
C
      CALL SEXPON (IS,T,NMAX)
      B = -A1
      V = 0.
      JMAX = NMAX+1
C
C

      DO 6 I=1,JMAX
C
C   HAVE NUMBER OF EVENTS EXCEEDED THE MAXIMUM NUMBER
C   THAT THE ARRAY CAN HOLD?
C
      IF (I.GT.NMAX) GO TO 8
      V = V+T(I)/((M-I+1)*B)
      IF (V.GT.TINT) GO TO 9
      T(I) = V
      IF (I.EQ.M) GO TO 10
    6 CONTINUE
C
C
C   NO EVENTS OCCURRED
C
    7 N = 0
      RETURN
C
C   TOO MANY EVENTS FOR ARRAY.  INCREMENT ERROR
C   CODE AND TRY AGAIN
C
    8 IER = IER+1
      GO TO 5
C
C   THE NUMBER OF EVENTS OBSERVED TO OCCUR IN THIS
C   NON-HOMOGENEOUS POISSON PROCESS IS 'N'
C
    9 N = I-1
      IF (N.EQ.0) RETURN
      GO TO 11
   10 N = M
   11 CONTINUE
C
C   IS THE RATE FUNCTION INCREASING OR DECREASING?
C
      IF (IFLAG.EQ.3) GO TO 13
C
C  TIME REVERSAL TECHNIQUE IS NECESSARY
C   DETERMINE WHETHER N IS EVEN OR ODD
```

```
C
      ISIG = MOD(N,2)
      NLOOP = N/2
      N1 = N+1
C
C
      DO 12 I=1,NLOOP
      S = T(I)
      T(I) = ER-T(N1-I)
      T(N1-I) = ER-S
   12 CONTINUE
C
C
      IF (ISIG.EQ.1) T(NLOOP+1)=ER-T(NLOOP+1)
C
C   ARE TIMES OF EVENTS REQUESTED?
C
      IF (II.EQ.0) RETURN
      GO TO 15
   13 IF (II.NE.0) GO TO 15
      IF (EL.EQ.0.) RETURN
C
C
      DO 14 I=1,N
      T(I) = EL+T(I)
   14 CONTINUE
C
      RETURN
C
C   CALCULATE TIMES BETWEEN EVENTS
C
   15 S = T(1)
C
C
      DO 16 I=2,N
      S1 = T(I)
      T(I) = T(I)-S
      S = S1
   16 CONTINUE
C
      RETURN
C
C   THE POISSON PROCESS IS HOMOGENEOUS
C
   17 I = 1
      U = 0.
      CALL SEXPON (IS,T,NMAX)
   18 U = U+T(I)
      IF (U.GT.PAR) GO TO 20
      I = I+1
      IF (I.GT.NMAX) GO TO 19
      GO TO 18
C
C   INCREMENT ERROR CODE
C
   19 IER = IER+1
C
C   TRY AGAIN WITH NEW STRING OF VARIATES
C
      GO TO 17
   20 N = I-1
      IF (N.EQ.0) RETURN
      IF (II.EQ.1) GO TO 22
C
C
      DO 21 I=1,N
      EL = EL+AINVRS*T(I)
      T(I) = EL
   21 CONTINUE
C
C
      RETURN
```

```
C
C
   22 DO 23 I=1,N
      T(I) = T(I)*AINVRS
   23 CONTINUE
C
      RETURN
      END
C    SUBROUTINE PVAR GENERATES A POISSON (THETA)
C    VARIATE, M, USING THE GAMMA METHOD
C
      SUBROUTINE PVAR (IS,THETA,M)
      DIMENSION T(5000)
      COMMON /HOLD/ T
      K = 0
      C = 16.0
      D = .875
    1 IF (THETA.LT.C) GO TO 2
      GO TO 5
    2 U = 1.
      CTN = EXP(-THETA)
      MMAX = THETA+6.*SQRT(THETA)
    3 I = 1
      CALL SRAND (IS,T,MMAX)
    4 U = U*T(I)
      IF (U.LT.CTN) GO TO 8
      I = I+1
      K = K+1
      IF (I.GT.MMAX) GO TO 3
      GO TO 4
    5 NP = INT(D*THETA)
      AN = FLOAT(NP)
      CALL GAMA (AN,IS,G,1)
      IF (G.GT.THETA) GO TO 6
      K = K+NP
      THETA = THETA-G
      GO TO 1
    6 L = THETA/G
      NP = NP-1
      CALL SRAND (IS,T,NP)
C
C
      DO 7 I=1,NP
      IF (T(I).LT.U) K = K+1
    7 CONTINUE
C
C
C
C    THE VALUE M IS ASSUMED BY THE POISSON (THETA) VARIATE
C
    8 M = K
      RETURN
      END
C    SUBROUTINE REJECT GENERATES AN ORDERED SAMPLE
C    OF GIVEN SIZE FROM THE SECOND COMPONENT
C    OF THE ORIGINAL INTENSITY FUNCTION
C    USING A REJECTION-ACCEPTANCE ALGORITHM
C
      SUBROUTINE REJECT (IS,EL,CSTAR,PVEC,Q,L)
      DIMENSION V(500), PVEC(5)
      DIMENSION T(5000)
      COMMON /HOLD/ T
      L20 = L*10
      IF (L20.GT.500) L20=500
      L1 = L+1
      K = 1
    1 J = 0
      CALL SRAND (IS,V,L20)
C
C
      DO 2 I=1,L20
      J = J+1
```

```
      T(K) = Q*V(J)+EL
      J = J+1
      IF (V(J).LT.CALC(PVEC,T(K))/CSTAR) K=K+1
      IF (K.EQ.L1) GO TO 3
      IF (J.GE.L20-1) GO TO 1
    2 CONTINUE
C
C
      IF (K.LT.L) GO TO 1
    3 CALL PXSORT (T,1,L)
      RETURN
      END
C   SUBROUTINE COLATE SUPERPOSES TWO ORDERED
C   EVENT STREAMS OVER THE SAME INTERVAL
C
      SUBROUTINE COLATE (LBGN,LFIN,JBGN)
      DIMENSION TIMES(5000), T(5000)
      COMMON /MIKE/ TIMES/HOLD/T
      I = JBGN
      J = I
      K = LBGN
    1 IF (TIMES(I).LT.TIMES(K)) GO TO 2
      T(J) = TIMES(K)
      J = J+1
      K = K+1
      IF (K.GT.LFIN) GO TO 3
      GO TO 1
    2 T(J) = TIMES(I)
      J = J+1
      I = I+1
      IF (I.EQ.LBGN) GO TO 5
      GO TO 1
    3 II = LBGN-1
C
C
      DO 4 N=I,II
      T(J) = TIMES(N)
      J = J+1
    4 CONTINUE
C
      RETURN
    5 CONTINUE
C
      DO 6 N=K,LFIN
      T(N) = TIMES(N)
    6 CONTINUE
C
C
      RETURN
      END
C   SUBROUTINE HELP EVALUATES THE INTEGRATED INTENSITY
C   FUNCTION OVER THE INTERVAL (EL,ER)
C
      SUBROUTINE HELP (A,A1,A2,EL,ER,XX)
      DOUBLE PRECISION MMDAW,BB,AA
      Z = SQRT(ABS(A2))
      Y = (A1*Z)/(2.*A2)
      AA = Z*EL+Y
      BB = Z*ER+Y
      CC = A-A1*A1/(4.*A2)
      CC = EXP(CC)/Z
      IF (A2.LT.0.) GO TO 1
      Q1 = DEXP(AA**2)*MMDAW(AA)
      Q2 = DEXP(BB**2)*MMDAW(BB)
      XX = CC*(Q2-Q1)
      RETURN
    1 CC = CC*.8862269
      XX = CC*(DERF(BB)-DERF(AA))
      RETURN
      END
C   FUNCTION CALC EVALUATES THE SECOND COMPONENT OF
C   THE DECOMPOSED INTENSITY FUNCTION
```

```
C    FOR ANY INPUT VALUE.
C
      FUNCTION CALC (P,ABSA)
      DIMENSION P(5)
      X = P(1)+P(2)*ABSA+P(3)*ABSA**2
      XX  = P(4) +P(5)*ABSA
      CALC =  EXP(X)-EXP(XX)
      RETURN
      END
```

LIST OF REFERENCES

1. Lewis, P.A.W., and Shedler, G.S., Simulation of Non-Homogeneous Poisson Processes by Thinning, Naval Postgraduate School, June 1978.

2. Lewis, P.A.W., and Shedler, G.S., Simulation of Non-Homogeneous Poisson Processes With Degree-Two Exponential Rate Function, IBM Research Division, 1977.

3. Patrow, M.L., A Comparison of Two Algorithms for the Simulation of Non-homogeneous Poisson Processes with Degree-Two Exponential Polynomial Intensity Function, Masters Thesis, Naval Postgraduate School, 1977.

4. IBM System/360 Operating System Supervisor Services and Macro Instructions (GC28-6646-7), 8th ed., p. 51, International Business Machines, 1974.

5. Naval Postgraduate School Report NPS55Lw73061A, Naval Postgraduate School Random Number Generator Package LLRANDOM, by G. Learmouth and P.A.W. Lewis, 1973.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	2
4.	Assoc. Professor A.F. Andrus, Code 55As Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
5.	Professor D.P. Gaver, Jr., Code 55Gv Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
6.	Assoc. Professor P.A. Jacobs, Code 55Jc Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
7.	Professor P.A.W. Lewis, Code 55Lw Department of Operations Research Naval Postgraduate School Monterey, California 93940	3
8.	CAPT M.L. Patrow, USMC 13119 Orleans Street Woodbridge, Virginia 22192	1
9.	Mr. G.S. Shedler IBM Research Laboratory IBM Building San Jose, California 95193	1
10.	Assoc. Professor A.R. Washburn, Code 55Ws Department of Operations Research Naval Postgraduate School Monterey, California 93940	1

		No. Copies
11.	LCDR J.S. Redd, USN 7313 Essex Avenue Springfield, Virginia 22150	2

180153
Thesis
R2653 Redd
c.1
Generation of non-homogeneous poisson processes by thinning: programming considerations and comparison with competing algorithms.

[illegible] NOV 79 25643
21 APR 80 25643
21 APR 80 25643
26 MAY 81 264051

180153
Thesis
R2653 Redd
c.1
Generation on non-homogeneous poisson processes by thinning: programming considerations and comparison with competing algorithms.

thesR2653
Generation of non-homogenous poisson pro
3 2768 001 01278 4
DUDLEY KNOX LIBRARY